DERRIDA'S MACHINES PART III

BYTES & PIECES

of

PolyLogics, m-Lambda Calculi,
ConTeXtures

ConTeXtures
Programming Dynamic Complexity*

© *by Rudolf Kaehr*
ThinkArt Lab Glasgow Hallowe'en 2005

"Interactivity is all there is to write about:
it is the paradox and
the horizon of realization."

Sponsored, partly, by the German Software ThinkTank **ALGoLL AG, Munich, Germany .*

ConTeXtures

Programming Dynamic Complexity

A. Designing ConTeXtures

B. General Stratagemes of ConTeXtures

C. Sketch of a General Theory of Subjectivity

D. Mapping ARS onto the PCL-Matrix

E. Mapping General Topics

Poly-Arithmetical Topics

List-Topics

Transjunctional Patterns

Permutational Patterns

Why not jump from Y to Why?

F. Poly-Topics

G. Dissemination of Programming Styles

ConTeXtures

Programming Dynamic Complexity*

© by Rudolf Kaehr

ThinkArt Lab Glasgow April 2005

Interactivity is all there is to write about:
It is the Paradox and
the Horizon of Realization."

Sketch of a new programming paradigm for epistemologically complex and dynamic situations. *ConTeXtures* risks a start with the insight of the relevancy of textuality in contrast to name- and statement-based approaches and the necessity of a multitude of logically and computationally relevant viewpoints in modelling and programming complexity. *ConTeXtures* are understood as a distribution and mediation of classical programming paradigms. Complexity is thematized as a constellation of interactivity and reflectionality which is not reducible to a single, mono-contextural, logico-computational process-oriented approach.

ConTeXtures are developed for conceptual programing and computational reality construction. Also it is not excluded, real-world problem solving is not yet in the main focus of this introduction to ConTeXtures.

ConTeXtures' emphasis is on interactionality / reflectionality is in a strong complementarity to the guiding philosophy of abstraction as it is proclaimed in the famous statement of Guy L. Steels *"Abstraction is all there is to talk about: it is the object and the means of discussion."*

Sponsored by the German Software ThinkTank **ALGoLL AG, Munich, Germany.*

Introduction

The aim of this text is to give a short introduction of the idea of polycontexturality and a sketch of mapping the programming paradigm and apparatus of the Lambda Calculus based A++ (ARS) (Georg Loczewski) onto polycontextural structures and strategies. Polycontexturality is not used a new singular meta-concept to deal with multi-centred complexity, but as a complexity in itself, which is realized as an interplay of its autonomous aspects involving the facets of multi-/dis-/trans-/polycontexturality. The result will be a sketch of a new way of programming called *ConTeXtures* which is build by a multitude of mediated classical Lambda Calculi distributed hierarchically in relation to reflectional and heterarchically in relation to interactional thematizations. As a notational system, also to visualize the conceptuality, the polycontextural *matrix* and its interpretation by the *bracket* method is introduced. The name *ConTeXtures* is based on contextures and inter-textual thematizations and the X of chiasm (proemial relationship).

Contents

1. The study gives a new introduction to the theory of polycontexturality emphasizing especially the features of reflectionality and interactivity between contextures building a general matrix of polycontexturality.

2. This polycontextural matrix gets a step wise concretization through a mapping of the programming paradigm A++ (ARS) onto it. This is producing a general theory of disseminated programming languages. The steps of modelling, producing the general tectonics of ConTeXtures, will be a mapping from *Matrix* to *Templates* to *Patterns* to *Configurations* to *Constellations*.

3. Some new features of ConTeXtures, developed by the strategy of the *"heterarchic cut"*, are introduced along the lines of prototypical applications of the topics (data types, sorts) and programming styles. This will include an example of heterarchy for OOP, distributed reflection for reflectional programming, heterarchic parallelism for symbolic and functional programming, and others.

4. Philosophical and grammatological backgrounds of ConTeXtures will be sketched and contrasted to the logocentric foundations of the Lambda Calculus, esp. in its form as ARS. More can be found in SKIZZE-0.9.5 (german) and DERRIDA'S MACHINES (english).

Other names for *ConTeXtures*: poly-A++, poly-ARS, SAMBA'S.

Historical Remarks

ConTeXtures are a further specification of the approaches proposed in SKIZZE-0.9.5 and DERRIDA'S MACHINES, PART I/II and are published as DERRIDA'S MACHINES PART III. I started this work on *ConTeXtures* just after having finished the "funky" collage/montage/sabotage SUSHI'S LOGICS at the end of 2004 where I mentioned the work of Georg Loczewski on his A++ (ARS). The main ideas to ConTeXtures goes back to 1986 when I tried to develop a *trans-FORTH* language for combined FORTH processor systems, later TRANSPUTERS, for modelling "living systems" supported by the Stiftung Volkswagen, Hannover, Germany. But failed to get further funding for this research. The project to develop a framework of *Dynamic Semantic Web* lost its funding, too. The interesting perspectives opened up there may have found a further realization in the framework of the non-funded anti-foundational paradigm of ConTeXtures.

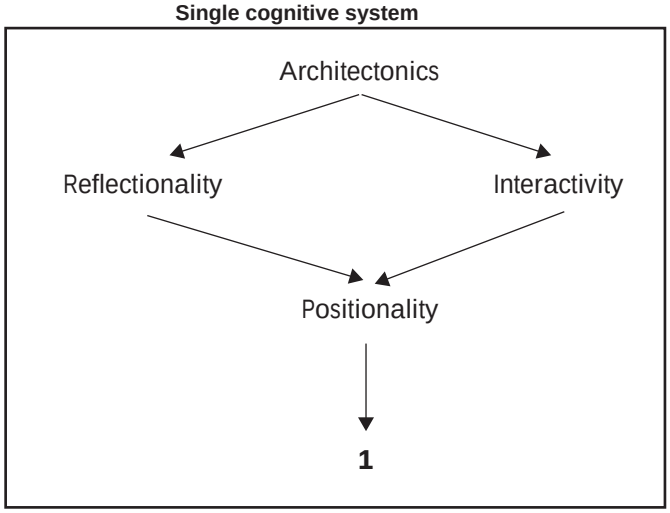
A. Designing ConTeXtures

1 General Model of Polycontexturality

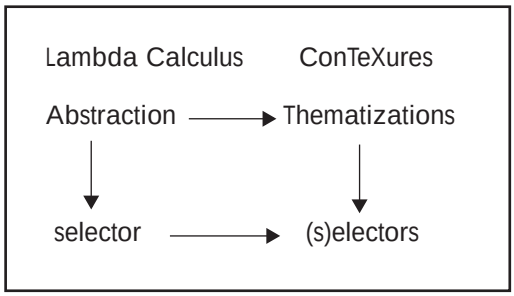
architextures enfold and unwrap spaces. As their outside, they constitute space itself. Though a-topic, architextures are the indispensable material counterpart to the immateriality of space. *architextures* are involved in the formation of all social, historical and cultural spaces. Sometimes as their hidden infrastructure, as the invisible order of signification; sometimes as their utmost visible dimension, as their bi-dimensional interface: architextures are essentially "*textures*", the weaved fabric of lines and voids, layers and surfaces. To follow the sinuous folds of these swathes, an archaeology of another kind is needed, an archeology of synchronism dealing with both past and present, an archeology of "synopsism" contemplating both surfaces and grounds.
http://www.atopiaonline.de/architex/architex.htm

Skizze eines Gewebes rechnender Räume in denkender Leere.
http://www.thinkartlab.com/pkl/media/SKIZZE-0.9.5-Prop-book.pdf

Diagramm 1 **Single cognitive system**



ConTeXtures are realizing the idea of thematization in developing a paradigm of a new programming system understood as a dissemination of classical paradigms. Programming languages based on the Lambda Calculus



Programming languages based on the Lambda Calculus are founded in the idea of *abstraction* and realized by the basic operator of *selection*. ConTeXtures are disseminating this idea and techniques as a step to realize the idea of *thematization* and its basic operators of *elections* of Lambda Calculus based languages.

B. General Stratagemes of ConTeXtures

2 Design approach

$$
\begin{array}{l}
sketch - horizon^{(m)} \\
\left[build - architectures \right. \\
\quad \left[thematize - szenario \right. \\
\quad\quad \left[identify - contextures \right. \\
\quad\quad\quad \left(define - operations \right. \\
\quad\quad\quad\quad \left(abstract - function \right. \\
\quad\quad\quad\quad\quad \left(\{ propose - statements \} \right) \right) \right] \right] \left. \right]
\end{array}
$$

sketch-horizons: The operator *samba* is sketching or designing the horizon of the computational constellation, that is, the general structure of the evoqued textuality under consideration. Its complexity is given first by the positive number m: samba$^{(m)}$. Second, it may also include the very fundamental structure or pattern of the horizon. It is crucial to know how the fundament of the architectonic is conceived or build. Its graph-combinatorics can be a constellation from a linear to a star order. Designing a horizon of programming is realizing the demand for reality construction by programming as a stronger approach than problem solving.

design-architectures: The super-operators are defining the interplay between different contextures of the designed scenario. The main modi of this interplay are *interactivity* and *reflectionality* and locally *iterativity* (computations). Other dimensions, like intervention and anticipation, are possible, but are not explicitly included in this sketch.

thematize-scenarios: *styles:* The different programming styles or paradigms, distributed over the mediated contextures, have to be addressed. The styles can be in a mono- or poly-style constellation. Styles are the paradigms of functional, imperative, object-oriented, contextural and reflectional programming.

topics: The different operators, distributed over the mediated contextures, have to by thematized, taken into consideration. The operation thematize is defining the topics (mono- and poly-topics) to be brought into the scope of programming.

identify-contextures: Single contextures with and without interactive and reflectional connections to other intra- and trans-contextures are focussed. Identify-contextures is a function which is decomposing the complexity of the polycontextural situation given by the thematizing operation into its intra- and trans-contextural parts or modules. To each isolated or interacting contexture corresponds intra-contexturally a Lambda Calculus based ARS system inheriting the reflectional and interactional distribution of the contextures.

local ARS systems:

define-operations: Define <Name of Abstraction>

abstract-functions: lambda <List of Parameters>

propose-statements: {Statements}

3 Operational approach

samba (architectonics (reflectionality (interactivity
(define (lambda (statements)))))

samba:

the general *epistemic activity* ("abstraction") of thematizing the world under consideration. Thematizing happens as plurality, there are always a multitude of viewpoints of thematizing. These viewpoints which opens up contextures are not isolated events, they are interacting and being together, they are mediated and the rules how to move from one point of view to another have to be given.

Thus samba is the operator of thematizing mediated complexity, that is, polycontexturality. The general rules of polycontexturality are introduced by the proemial relationship. SAMBA'S as a dynamic programming (con)texture is involving operators of evolution and emanation, EVOL and EMAN, to model structures of "living tissue". In this study we are considering only stable systems, that is, EVOL and EMAN are, at the time, not applied. Samba is opening up the horizon of thematization. It has the function of an epistemological "Entwurf", design. A design can be to broad or it can be to narrow. Samba is flexible to adapt to the complexity / complication of the needed design.

Also the operator *samba* is the start decision of a specific programming, it can start everywhere inside of the design and programming. Samba is self-applicable in an intra- and a trans-contextural sense.

Global characteristics:

architectonics:

complexity and structure of disseminated systems A++.

Architectonics is the process of creating the architecture of a system. Architectonics applies to the polycontexturality of the disseminated systems. In contrast, tectonics is applied to the intra-contextural architecture or morphology of the distributed formal systems. The dissemination of formal systems like A++ can be monomorphic or polymorphic. Monomorphic dissemination is distributing and mediating systems with equal architectonics, polymorphic dissemination is distributing and mediating systems with different tectonics. Architectonics is considering the distribution of *"logical loci"*.

reflectionality:

mechanism and degree of modeling outer or inner neighbor systems into local systems from simple mirroring to complex metamorphosis. To mirror another system or other processes by the reflection operator is to repeat (replicate, clone) it into an inner environment of the mirroring system. The inner environment gives the space for co-existence of the repeated system and its processes without overriding the mirroring system. The inner locus for replication of the other system is given, that is produced by the architecture of the whole system.

Another kind of reflectionality is enhancing the complexity of the system, say from S1 and S2 to S3 as a reflecting and mediating system. The possibility of inner environments is also giving logical loci for meta-reflection and introspection in the sense of Gunther's Inro-Semantics.

interactivity:

Super-operators are managing interactivity between disseminated systems without metamorphosis. This includes simultaneity, like for transjunctions, and replications.

Local Characteristics:

identify and thematize:

identify the contexture(s) under consideration out of the whole complexity.

Selects a starting point of calculation in a given complexity.

define:

local *define* in two configurations:

Intra-contextural local define

Trans-contextural local define

lambda:

local *lambda* in two configurations:

Intra-contextural local lambda

Trans-contextural local lambda

statements:

Statements in SAMBA are contextual and not isolated sentences. SAMBAS is (con)text based and SAMBAS sentences are inter-textually defined in contrast to the sentence and name based LAMBDA where sentences or statements are isolated units constructed by atomic terms and connectors to construct composed statements. Cf. http://www.thinkartlab.com/pkl/lola/poly-Lambda_Calculus.pdf

3.1 Comments

Global and local references

The main reference of ConTeXtuers are reflectional and interactional patterns of the different contextures' interplay and not the topics (data types) of the systems involved which are the references of their statements. They may be called global references in contrast to the local references of the local ARS systems. In general, references may be isomorphic, supporting the identity of the referred objects, or metamorphic, transforming their categorial identity. Metamorphic actions are ruled by the as-category of thematization allowing unrestricted referentiality over all levels of tectonics of ConTeXtures without confusion and attempts to connect everything with everything.

Hierarchy and heterarchy

This hierarchical order of brackets or functions shouldn't mislead to be read as the genuine structure of the programming texture. A second step of exposing the computational structure of ConTeXtures is given by the introduction of the *heterarchy diagram*. The head of ConTextures is a heterarchy which is ruling the multitude of interacting and reflecting hierarchical ARS-systems building the body of a societal constellation.

Why A++ (ARS)?

We could take any other programming language to involve into contextural distribution and mediation. It seems that A++ is very basic but nevertheless allows to develop prototypes of programs in different programming styles. ARS (A++) is not too radically reduced like the totally minimalist approaches of Iota/Jota and is equally much more to the point than Scheme. Its clear and honest introduction/invention by Georg Loczewski played a significant, motivational and inspirational, part to my own constructions.

3.2 General Bracket-Scheme of ConTeXtures

Diagramm 2 **Global general scheme for ConTeXtures**

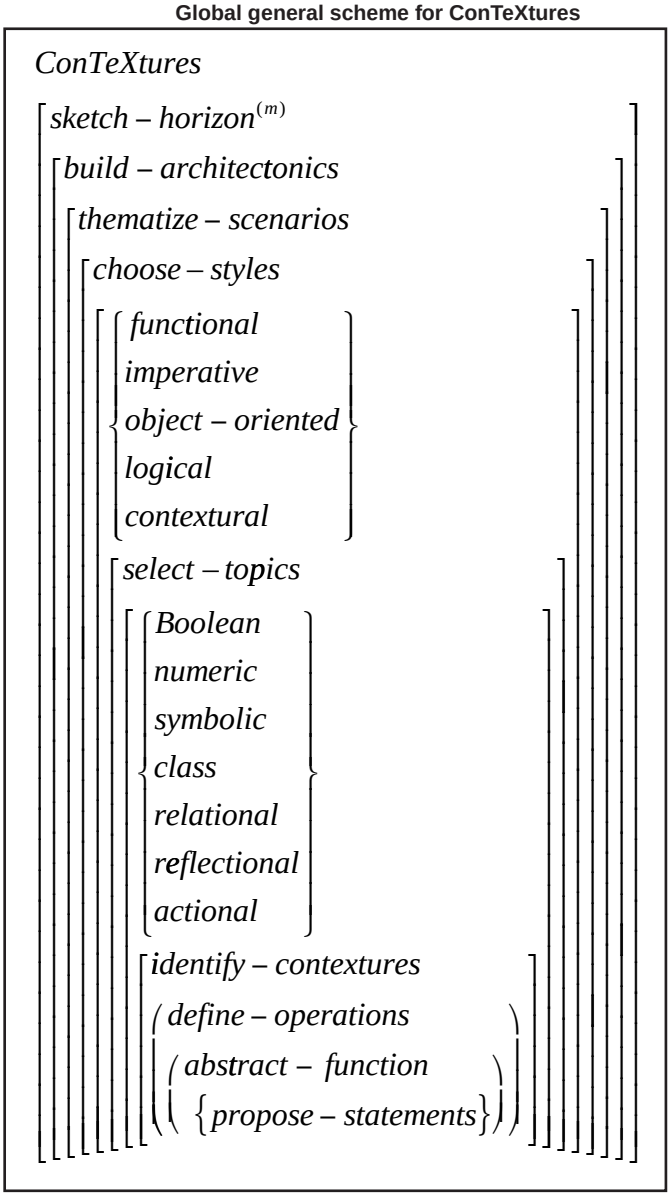


Diagramm 3 **Local general scheme for A++**

$$General\ Scheme\ for\ A++$$
$$\left(\begin{array}{l} define < Name\ of\ Abstraction > \\ \left(\begin{array}{l} lambda < List\ of\ Parameters > \\ \left\{\begin{array}{l} < Statement_1 > \\ < Statement_2 > \\ \quad \cdots\cdots \\ < Statement_n > \end{array}\right. \end{array}\right) \end{array}\right) \qquad \left[\begin{array}{l} contexture \\ \left(\begin{array}{l} head \\ (body) \end{array}\right) \end{array}\right]$$

3.2.1 Contextural Dynamics in the General Bracket-Scheme of ConTeXtures

The local scheme of A++ has to be interwoven with its neighbor local systems involving some contextualizations of the basic local definitions which are not made explicit in the General Bracket Scheme but are a natural consequence of the head structure of the design. Contextualizations of concepts are realized mainly by the AS-category. In ConTeXtures, the as-category appears as an abstraction, the *as-abstraction*. To thematize is always to thematize something *as* something. The as-category is involving the whole conceptuality of ConTeXtures and has to be realized on all levels of its tectonics (design, thematize, identify, define, topics, styles).

For introductionary reasons I have to restrict my representation of ConTeXtures and these interesting aspects of the structural *dynamics* of ConTeXtures, produced by the as-abstraction based on proemiality, to a *stable* exposition. Dynamics will be of some significance, later, for the concept of "metamorphic reflection" and the process of heterarchization of OOP concepts.

Diagramm 4 **Contextualized "Name of Abstraction"**

$$General\ Scheme\ for\ A++\ in\ poly-A++$$
$$\left(\begin{array}{l} define < (Name\ of\ Abstraction)_i\ AS\ (Name\ of\ Abstraction)_j > \\ \left(\begin{array}{ll} \left(\begin{array}{l} lambda < List\ of\ Parameters > \\ \left\{\begin{array}{l} < Statement_1 > \\ < Statement_2 > \\ \quad \cdots\cdots \\ < Statement_n > \end{array}\right. \end{array}\right) & \left(\begin{array}{l} lambda < List\ of\ Parameters > \\ \left\{\begin{array}{l} < Statement_1 > \\ < Statement_2 > \\ \quad \cdots\cdots \\ < Statement_n > \end{array}\right. \end{array}\right) \end{array}\right) \end{array}\right)$$

Reduction of the as-abstraction to the identity abstraction

define name$_i$ as name$_j$: for i=j, this construct reduces to the classic: *define name.*

The as-abstraction: The name X as the name Y is the name Z.

The name-abstraction: the name X as name X is name X.

4 General Mapping Strategies

GM: Matrix –> Templates –> Patterns –> Configurations –>Constellations

PM	$O1$	$O2$	$O3$
$M1$	$\varnothing$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$\varnothing$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$S_{1.2}$	$S_{1.3}$
$M2$	$S_{2.1}$	$\varnothing$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$S_{2.1}$	$S_{3.1}$
$M2$	$S_{1.2}$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$S_{2.3}$	$S_{3.3}$

1. Matrix

General *complexity* and *complication* of the balanced matrix (m=n) with its combinatorics of abstract positions as abstract place holders for ARS systems.

2. Templates

Different *reflectional* and *interactional* combinations of contexture based ARS systems S_i in a balanced positional matrix.

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_1	S_2	$\varnothing$
$M3$	S_1	$\varnothing$	S_3

3. Patterns

Mapping the super-operators {id, perm, red, bif, repl} on the templates as horizon.

$samba^{(3)}(repl, id, id) - horizon$

$$\begin{bmatrix} thematize\ (topics^{(3)}) \\ \begin{bmatrix} \begin{bmatrix} identify\ contextures \\ \begin{bmatrix} define\ name \\ \begin{bmatrix} \begin{pmatrix} lambda_{1.1} \\ \{statements\} \end{pmatrix} \\ \begin{pmatrix} lambda_{1,2} \\ \{statements\} \end{pmatrix} \\ \begin{pmatrix} lambda_{1,3} \\ \{statements\} \end{pmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} & \begin{bmatrix} identify\ contexture_{2.2} \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} & \begin{matrix} \varnothing \\ \varnothing \\ \begin{bmatrix} identify\ contexture_{3.3} \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} \end{matrix} \end{bmatrix} \end{bmatrix}$$

4. Configurations

Combinations of the different *topics* (Numeric, Boolean, Symbolic, etc.) on the patterns. And the combinations of different programming *styles*.

5. Constellations

Combinations of the concrete realizations of topics as operators and operands in homogene or heterogene programming styles.

4.1 Mapping general templates unto the matrix

PM	$O1$	$O2$	$O3$
$M1$	$\varnothing$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$\varnothing$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$\varnothing$

This kind of mapping is still, more or less, neutral to the difference of reflectionality/interactionality and is marking only the occupancy of the loci by a subsystem and defining its complexity, m=3, and its type as balanced, in contrast to over- or under balanced matrices.

4.1.1 Positional matrix

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$S_{1.2}$	$\varnothing$	$\varnothing$
$M3$	$S_{1.3}$	$\varnothing$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	$S1$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$S2$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$S3$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$S_{2.1}$	$\varnothing$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$\varnothing$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$S_{2.2}$	$S_{2.3}$
$M3$	$\varnothing$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$\varnothing$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$S_{3.2}$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$S_{2.1}$	$S_{3.1}$
$M2$	$S_{1.2}$	$\varnothing$	$\varnothing$
$M3$	$S_{1.3}$	$\varnothing$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$S_{1.2}$	$S_{2.2}$	$\varnothing$
$M3$	$S_{1.3}$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$S_{1.2}$	$S_{1.3}$
$M2$	$S_{2.1}$	$\varnothing$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$S_{1.2}$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$S_{1.3}$
$M2$	$\varnothing$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$S_{3.2}$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$S_{1.3}$
$M2$	$\varnothing$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$S_{3.2}$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$S_{2.1}$	$\varnothing$
$M2$	$S_{1.2}$	$S_{2.2}$	$\varnothing$
$M3$	$S_{1.3}$	$S_{2.3}$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$S_{1.3}$
$M2$	$S_{2.1}$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$S_{3.2}$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$S_{2.1}$	$S_{3.1}$
$M2$	$S_{1.2}$	$S_{2.2}$	$S_{2.3}$
$M3$	$S_{1.3}$	$S_{2.3}$	$S_{3.3}$

4.1.2 Templates unto the matrix

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1}$	$\varnothing$	$\varnothing$
$M2$	$S_{1.2}$	$S_{2.2}$	$S_{3.2}$
$M3$	$\varnothing$	$\varnothing$	$S_{3.3}$

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_1	S_2	S_3
$M3$	$\varnothing$	$\varnothing$	S_3

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_2	S_2	S_2
$M3$	$\varnothing$	$\varnothing$	S_3

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_1	S_2	S_2
$M3$	$\varnothing$	$\varnothing$	S_3

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_2	S_2	S_3
$M3$	$\varnothing$	$\varnothing$	S_3

The positional matrix is interpreted by different templates.

4.1.3 Bracket exposition of constellations of templates

$$(O1O2O3)$$

$$\left[\left[\begin{matrix} O1 \\ \left(\begin{matrix} M1M2M3 \\ (G110) \end{matrix}\right) \end{matrix}\right]\left[\begin{matrix} O2 \\ \left(\begin{matrix} M1M2M3 \\ (G020) \end{matrix}\right) \end{matrix}\right]\left[\begin{matrix} O3 \\ \left(\begin{matrix} M1M2M3 \\ (G033) \end{matrix}\right) \end{matrix}\right]\right]$$

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_1	S_2	S_3
$M3$	$\varnothing$	$\varnothing$	S_3

Constellations of templates visualized by brackets. The advantage of the bracket method to the matrix method is its better operativity and visualization for complex formulas.

4.1.4 Fractalized constellations

$$(O1O2O3)$$

$$\left[\begin{array}{l}\left[O1\ \left(M1M2M3\ (G111)\right)\right]\\ \left[O2\ \left(M1\left(M2\left(M3\ (G003)\right)\right)\ (G100)\ (G222)\right)\right]\\ \left[O3\ \left(M1M2M3\ (G033)\right)\right]\end{array}\right]$$

PM	$O1$	$O2$	$O3$
$M1$	S_1	$S_{2.1}$	$\varnothing$
$M2$	S_1	$S_{2.0}$	S_3
$M3$	S_1	$S_{2..3}$	S_3

$$(O1O2O3)$$

$$\left[\begin{array}{l}\left[O1\ \left(M1M2M3\ \left(G110\right)\left(M1\ \left(G010\right)\left(M1\ \left(G010\right)\left(M1\ \left(G010\right)\right)\right)\right)\right)\right]\\ \left[O2\ \left(M1M2M3\ (G222)\right)\right]\\ \left[O3\ \left(M1M2M3\ (G033)\right)\right]\end{array}\right]$$

PM	$O1$	$O2$	$O3$
$M1$	S_1	S_2	$\varnothing$
$M2$	$S_{1.1.1.1}$	S_2	S_3
$M3$	$\varnothing$	S_2	S_3

Constellations of templates can naturally be iterated producing some iterative, recursive and fractal structures of the templates, visualized by the matrix as well the bracket method. An alternative presentation is to localize contextural modules at their loci in a diagram.

4.2 General patterns of templates

Interpretation of the distribution of subsystems, general patterns, in the general matrix of templates by super-operators.

$(O1O2O3)$

$$\left[\left[\begin{matrix} O1 \\ \left(\begin{matrix} M1 \\ (G110) \end{matrix}\right) \end{matrix}\right]\left[\begin{matrix} O2 \\ \left(\begin{matrix} M1M2M3 \\ (G020) \end{matrix}\right) \end{matrix}\right]\left[\begin{matrix} O3 \\ \left(\begin{matrix} M1M2M3 \\ (G023) \end{matrix}\right) \end{matrix}\right]\right]$$

$S_{110}S_{020}S_{033}$: (repl, bif, repl)

S120S020S023 : (id, bif, id)

S110/020S020S023 : (repl, bif, id)

S110S020S023 : (repl, bif, id)

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	$S_{1.2}$	S_2	S_2
$M3$	$\varnothing$	$\varnothing$	S_3

$(O1O2O3)$

$$\left[\left[\begin{matrix} O1 \\ \left(\begin{matrix} M1M2M3 \\ (G110) \end{matrix}\right) \end{matrix}\right]\left[\begin{matrix} O2 \\ \left(\begin{matrix} M1M2M3 \\ (G020) \end{matrix}\right) \end{matrix}\right]\left[\begin{matrix} O3 \\ \left(\begin{matrix} M1M2M3 \\ (G033) \end{matrix}\right) \end{matrix}\right]\right]$$

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_1	S_2	S_3
$M3$	$\varnothing$	$\varnothing$	S_3

4.2.1 A realization of a pattern of templates in ConTeXtures: (repl, id, id)

$$
\begin{array}{l}
samba^{(3)}(repl,\ id,\ id) - horizon \\
\left[\begin{array}{l}
thematize\ (topics^{(3)}) \\
\left[\begin{array}{l}
identify\ contextures^{(3)} \\
\left[\begin{array}{l}
identify\ contexture_{1} \\
\left[\begin{array}{l}
define\ name \\
\left[\left(\begin{array}{l} lambda_{1.1} \\ \{statements\} \end{array}\right)\right] \\
\left[\left(\begin{array}{l} lambda_{1,2} \\ \{statements\} \end{array}\right)\right] \\
\left[\left(\begin{array}{l} lambda_{1,3} \\ \{statements\} \end{array}\right)\right]
\end{array}\right]
\end{array}\right]
\left[\begin{array}{l}
identify\ contexture_{2.2} \\
\left[\left(\begin{array}{l} define\ name \\ \left[\left(\begin{array}{l} lambda \\ \{statements\} \end{array}\right)\right] \end{array}\right)\right]
\end{array}\right]
\left[\begin{array}{l}
identify\ contexture_{3.3} \\
\left[\left(\begin{array}{l} define\ name \\ \left[\left(\begin{array}{l} lambda \\ \{statements\} \end{array}\right)\right] \end{array}\right)\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}
$$

5 Thematizing: Abstracting the processuality of abstractions

5.1 Linearity, tabularity and (s)electors

A++ has as its presupposition the linear order of its basic terms. This linearity, based on the fundamental laws of *logos*, is behind the introduction of the selector function "sel". This introduction is conventional, referring to the culture of logocentrism and its linearity. In contrast, poly-A++ is based on the chiastic structure of its disseminated systems, therefore the conventional introduction of order, exchange and coincidence relations are behind the definitions of the systems objects, like truth values.

The IF- Abstraction

One of the most basic operation in any programming language is to make a decision, to select a block of code depending on the truth value of a certain argument.
Such an operation people have in mind, when they talk about IF-statements, IF-THEN-ELSE-constructs, alternative structures and a few more.
We can very well say that the IF-statement has the function to select code to be evaluated or executed. The IF-statement therefore is a function taking three arguments:

1.a condition having a certain truth value (true or false),
2.the first block of code and
3.the second block of code.

Because the selection of the code block to be evaluated or executed depends on the first argument, we can look at the first argument as a selector.

Having thus analyzed the essence of an IF-function we may code it as a `lambda abstraction':

```
(define if ( lambda ( a  b)
                  ( sel a b )))
```

Body of the IF-Abstraction

The body of this lambda abstraction is just a synthesis of the selector with its two arguments, the two blocks of code from which to select.
Because there are only two possibilities to perform a selection we distinguish between two selectors:
1.The selector which selects the first argument we assign the name `true'
2.and the other selector, which selects the second argument, we give the name `false'.

The lambda-abstractions for true and false are easily written as follows:

```
(define true (lambda (a b )
                          a))
(define false (lambda (a b )
                          b))
```

http://www.aplusplus.net/bookonl/node74.html

5.2 Selectors and electors: Intra-, trans- and poly-contextural selectors

ConTeXtures as poly-A++ is based not on linearity but on a more complex topology, here, on a tabular order to provide the chiastic distribution of different A++-systems.

As a consequence, the basic abstraction of selection "sel" has to be enhanced to a tabular selector, selecting at once terms from different neighbor systems. Terms thus, are not only intra-contextural in a linear order with predecessor / successor but globally in a tabular order with neighbor terms, too.

$$\left[\begin{array}{l} identify\ contexture_1 \\ \left(\begin{array}{l} define\ elect_1 \\ \left(\begin{array}{l} lambda\ (\ contextures^{(3)}\ ) \\ \quad elect\ \ contexture_2 \end{array}\right) \end{array}\right) \end{array}\right]$$

Because of the graphematical tabularity of ConTextures there are not *"only two possibilities to perform a selection"*. For each contexture there are intra-contexturally only two possibilities to perform a selection. But between contextures, trans-contexturally, there are as many new selectors as neighbor contextures. These new "selectors" should be called *electors.* Electors are electing the election for selectors to perform mutually each its selection.

That is, a selection can happen at once at different loci of a disseminated system. In other words, such a general a selector as any other successive or procedural action has to be realized in the two dimensions of intra- and trans-contexturality. Thus, a selector as a single operation can be realized intra-contextural staying in the same contexture or trans-contextural switching to another neighboring contexture. A selector as a complexion can be realized at once in different contextures. It could therefore be called a "poly-selector". Such a poly-selector can be defined as an overlapping of an intra-contextural selector "sel" and a trans-contextural selector, called *elector* "elect".

$$\left[\begin{array}{l} identify\ contexture_1 \\ \left(\begin{array}{l} define\ sel_1 \\ \left(\begin{array}{l} lambda\ (\ contextures^{(3)}\ ) \\ \quad elect\ \ contexture_1 \end{array}\right) \end{array}\right) \end{array}\right]$$

Thus, samba (elect, 1) = (sel).

The elector "elect" is (s)electing the contexture in which a selector "sel" is selecting its linearly ordered atomic terms and elements.

Selectors are acting in a pre-given order, they are conventionally introduced, not produced, but inherited from logocentric semiotics. Electors are involved in the evocation of new orders, new contextures, to give space for distributed selectors, positioning them into the graphematic matrix. Distributed selectors are constructed in the graphematic play of (s)electors and are not inherited from logocentrism.

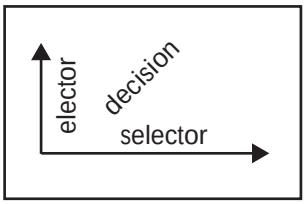
Electors and abstractions

The process to elect contextures has to be understood as a generalization of the process of naming. Naming is intra-contextural, and in ARS, it is quite general, allowing to name the process of naming itself, at least in some restricted sense. But it is not possible within the scope of this generalized concept of abstraction and naming to identify the frame of naming itself, and to name, call and identify it. This, in the rationality of strict identity of terms would automatically lead to circularity and self-destruction.

The process of electing contextures inside a contexture is possible only within the framework of sameness as contrasted to identity and diversity. Thus, a "self-call" is possible without paradoxes because it calls itself in the mode of sameness and not in the mode of identity. Electors are evoquing other contextures as the same con-

textures marking the other contexture and the othereness of contextures.

Contextural decisions are tabular with one dimension involving *selectors* the other dimension *electors*. Each posittion in ConteXtures is defined simoultaneously by its elecotrs and its selectors. There are no selectors without electors and there are no electors without selectors; both are designing the field of polycontextural reasoning and computing.



$$\left[\begin{array}{l} identify\ contextures^{(m)} \\ \left[\begin{array}{l} define\ elector \\ \left[\begin{array}{l} lambda\ (\ contextures^{(m)}\ ) \\ \left[\begin{array}{l} elect\ \ contexture \\ \left(\begin{array}{l} define\ selector \\ \left(\begin{array}{l} lambda\ (\ a\ b\ ) \\ (\quad\ (\ sel\ a\ b\ ) \end{array}\right) \end{array}\right) \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

An important circularity, or not?

As we can see here again, beginnings in logocentric systems are always circular. The if-function is defined by true/false of the selector and the abstractions true/false are defined by the two functions or meanings of the selector. That is, *the selector is defined by true/false* and *true/false is defined by the selector*.

5.3 Electors in different topologies

As mentioned before ConTeXtures are designed a long the distinction of interactionality and reflectionality, that is, a kind of a 2-dimensional dissemination of computational systems. This restriction happens for introductory reasons. There are no reasons why the abstraction of election has to be restricted to 2-dimensionality. In other words, the domain of the electors are defined by the type of dissemination. Dissemination can be realized in different topologies producing complex architectonics of polycontexturality.

5.4 Closure property of ARS and Closures in ConTeXtures

ARS (ARS) = ARS

$$\begin{array}{l} thematize\ (\ sel\ ) \\ \left(\begin{array}{l} define\ sel \\ \left(\left(\begin{array}{l} lambda\ (\ contextures^{(3)}\ ) \\ \left(\begin{array}{l} elect\ contexture_1 \\ \qquad\qquad \left(\begin{array}{l} lambda\ (\ a\ b\ ) \\ \qquad (\ sel\ a\ b\ ) \end{array}\right) \end{array}\right) \end{array}\right)\right) \end{array}\right) \end{array}$$

$$\left(\begin{array}{l} define\ sel \\ \left(\left(\begin{array}{l} identify\ contexture_1 \\ \left(\begin{array}{l} elect\ contexture_1 \\ \left(\begin{array}{l} lambda\ (\ a\ b\ ) \\ \qquad (\ sel\ a\ b\ ) \end{array}\right) \end{array}\right) \end{array}\right)\right) \end{array}\right) \quad \left(\begin{array}{l} define\ sel \\ \left(\left(\begin{array}{l} identify\ contexture_1 \\ \qquad \left(\begin{array}{l} lambda\ (\ a\ b\ ) \\ \qquad (\ sel\ a\ b\ ) \end{array}\right) \end{array}\right)\right) \end{array}\right)$$

To identify is to elect "inside" a thematization. To elect is leaving the contexture for another contextures of the compound.

$$\begin{array}{l} thematize\ (\ sel\ ) \\ \left[\begin{array}{l} lambda\ (\ contextures^{(3)}\ ) \\ \left[\begin{array}{l} elect\ contexture_1 \\ \left(\left(\begin{array}{l} define\ sel \\ \qquad \left(\begin{array}{l} lambda\ (\ a\ b\ ) \\ \qquad (\ sel\ a\ b\ ) \end{array}\right) \end{array}\right)\right) \end{array}\right] \end{array}\right] \end{array}$$

C. Sketch of a General Theory of Subjectivity

6 Cognition and Volition

Gunther's theory of subjectivity is introduced by the proemiality of cognition and volition distributed over the positions of I-subjectivity and Thou-subjectivity in common co-created environments.

In this study I interpret the terms cognition and volition by the terms reflectionality and interactivity.

Following the DiamondStrategies the duality of reflectionality and interactivity has to by dynamized by a first step to the quadruple:

6. Reflection as reflection: reflective reflection (short: reflection)
7. interaction as interaction: interactive interaction (short: interaction)
8. Interaction as reflection: reflectional interactivity (short: intervention)
9. Reflection as interaction: interactional reflectionality (short: interlocution)

If we would start the prospect with intervention / anticipation the catalogue would be:

10. Intervention as intervention
11. Intervention as anticipation
12. Anticipation as anticipation
13. Anticipation as intervention

Because no instance is primordial and rooting the chiasm all combinations of the 4 introduced possibilities have to be involved in the chiastic game of mutual founding.

Reflectionality is connected with the ability of an agent to iterate its inner environment placed at an architectonic locus into itself. In this sense reflectionality is a self-reflectional action not involving the neighbor agent in a behavioral sense. To model reflectionality in the context of the *founding relation* (Gunther) is supporting this self-referential understanding of reflectionality.

In contrast, interactivity is the ability of an agent to interact with its neighbor agent. To model interactivity in the context of the *proemial relationship* (Gunther) is supporting this approach to interactivity. The proemial relationship is ruling the possible modi of interaction and transformation (metamorphosis) between actors.

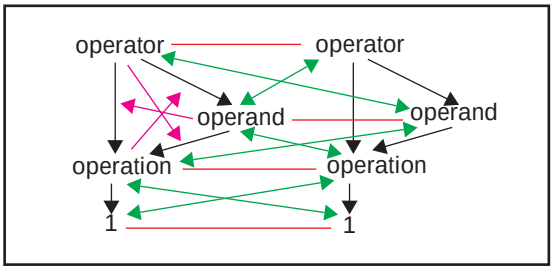
Limits

This paper is restricted to the study of the interplay of *reflectionality* and *interactivity* in computational systems in a very "harmonized" setting.

It is beyond the scope of this study to compare the concept of subjectivity with its treatment in philosophy of mind, psychology, cognitive sciences and reflectional programming, computational reflection, and others. This is left for another study.

7 Reflectionality

7.1 Operational introduction



Reflectionality comes into the general game if we thematize the relationality or operativity of the proemial construction from the point of view of an *internal* description / construction. An internal description has to consider all given concepts of a construction and to re-construct the builded construction out from the inside. An *external* description is realized by an external observer of the construction knowing the rules of construction. A full polycontextural description has furthermore to take into account the complementarity of internal and external descriptions of its constructions.

It reads as follows:

the operationality between operator and operand from the view of the operation,

the operationality between operator and operation from the view of the operand,

the operationality between operand and operation from the view of the operator.

And:

the operationality between operator and operand from the view of the position,
the operationality between operator and operation from the view of the position,
the operationality between operand and operation from the view of the position.
And so on.

The method of internal re-construction of the whole leads, conceived from a formal aspect, to the so called *context-valued logics* (Gunther 1968, Kaehr 1976) introducing the notion of logical invariance complementary to the notion of logical equivalency not yet been taken into account by the *observer theory* of second-order cybernetics.

Gunther's *"founding relation"* seems to be a complementary concept to the proemial relationship as interactivity and reflectionality appears to be complementary like the difference of internal / external descriptions and constructions.

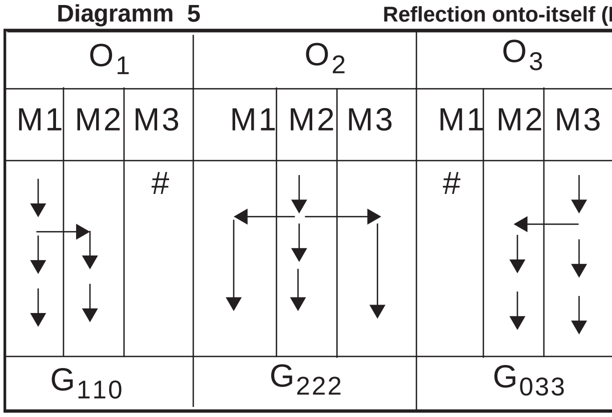
> This investigation intends only to show that the concept of Totality or Ganzheit is closely linked to the problem of subjectivity and trans-classic logic and that it is based on three basic structural relations:
>
> an exchange relation between logical positions
> an ordered relation between logical positions
> a founding relation which holds between the member
> of a relation and a relation itself.
>
> Thus we may say: *the founding-relation is an exchange-relation based on an ordered-relation. But since the exchange-relations can establish themselves only between ordered relations we might also say: the founding-relation is an ordered relation based on the succession of exchange-relations.* When we stated that the founding-relation establishes subjectivity we referred to the fact that a self-reflecting system must always be: self-reflection of (self-and hetero-reflection).
> Gotthard Günther, *Formal Logic, Totality and the Super-additive Principle*
> in: Beiträge zur Grundlegung einer operationsfähigen Dialektik, Band 1,
> Meiner Verlag, Hamburg, 1976, p.329-351, first publ.: BCL Report, 1966

7.1.1 Procedural introduction

The conceptual guiding metaphor may be "Reflexion-in-sich", reflection-into-itself of a the process of thematization. This has to be strictly distinguished from the principle of iteration and recursion of formulas or of the hierarchy of meta-levels or meta-languages in reflectional programming.

Also the design of ConTeXtures is in many ways in the tradition of Gunther's polycontextural logic our development feels nevertheless not restricted in any sense by this connection in its further developments.

Diagramm 5 **Reflection onto-itself (Reflexion in sich)**



$$(O1O2O3)\quad \left[\begin{array}{l} \left[\begin{array}{l} O1 \\ \begin{pmatrix} M1M2M3 & \\ & (G110) \end{pmatrix} \end{array}\right] \\ \left[\begin{array}{l} O2 \\ \begin{pmatrix} M1M2M3 & \\ & (G222) \end{pmatrix} \end{array}\right] \\ \left[\begin{array}{l} O3 \\ \begin{pmatrix} M1M2M3 & \\ & (G033) \end{pmatrix} \end{array}\right] \end{array}\right]$$

PM	$O1$	$O2$	$O3$
$M1$	S_1	S_2	$\varnothing$
$M2$	S_1	S_2	S_3
$M3$	$\varnothing$	S_2	S_3

It is of general importance to see that all reflectional but also all interactional procedures are rooted in their local beginning, their start of a deriviation or of a formula development. This is very obvious for logical proofs based on tableaux methods where attributes and signatures of trees are considered.

Iteration of Distinctions

Not everyone is a mathematician or a programmer, therefore it has to be assured that there are surely no limits in iterating introduced concepts, distinctions and constructions without any limits.

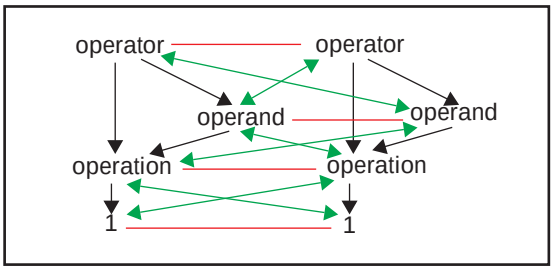
Iterative diagrams are not very practical, formulas and even matrices are more comprehensible.

Another approach to *structural iteration*, not studied in this introduction, would be the iteration of the inner arena of the scenario, that is, the iteration of the M-dimension while keeping the O-dimension constant, producing therefore overbalanced systems with m>n.

$$(O1O2O3)\quad \left[\begin{array}{l} \left[\begin{array}{l} O1 \\ \begin{pmatrix} M1M2M3 & \\ (G110) & \begin{pmatrix} M1M2M3 & \\ (G110) & \begin{pmatrix} M1M2M3 & \\ (G110) & \begin{pmatrix} M1M2M3 & \\ & (G110) \end{pmatrix} \end{pmatrix} \end{pmatrix} \end{pmatrix} \end{array}\right] \\ \left[\begin{array}{l} O2 \\ \begin{pmatrix} M1M2M3 & \\ & (G222) \end{pmatrix} \end{array}\right] \\ \left[\begin{array}{l} O3 \\ \begin{pmatrix} M1M2M3 & \\ & (G033) \end{pmatrix} \end{array}\right] \end{array}\right]$$

PM	$O1$	$O2$	$O3$
$M1$	$S_{1.1.1.1}$	S_2	$\varnothing$
$M2$	$S_{1.1.1.1}$	S_2	S_3
$M3$	$\varnothing$	S_2	S_3

8 Interactivity



Interactivity, which is not changing the structure of architectonics, can be seen as a kind of reflectionality, reflection-onto-others. In other words, with a stable architectonics which is excluding metamorphosis and evolution / emanation, both concepts are complementary. That is, reflectionality can be seen as an interactivity in the modus of replication into itself. Both activities are complementary to each other and have to be distinguished properly. In polycontextural logic interactivity is mainly realized by different kinds of *transjunctions*. But interactivity is a general concept and is not reduced to logical operations only. In the terminology of super-operators of ConTeXtures interactivity is represented by the operator "*bifurcation*".

Interactivity is not based on communication as information processing or message passing (Paul Dourish). It can be described as the action of addressing an addressee which is able to accept the addressing by its own addressable structure. After having been addressed and the addressing is accepted by the addressed and the addresser has recognized the acceptance of being addressed and the addressing is thus established, information can be exchanged between agents in the sense of processual communication (MAS, MIC).

Interaction is not information or communication

Interaction is action and not information. Informations maybe special actions. Interactivity in the sense of PCL is understood as a kind of action. Therefore, to use semiotic terms, it is a genuine pragmatic function. But pragmatics is still not well developed and shouldn't be based on linguistic paradigms. In a more radical terminology interactivity (as well as reflectionality) should be understood as based in the *togetherness* (Mitsein, Heidegger) of agents ("embodied interaction", Dourish) and focussed not on "*what*" is processed (information) but "*how*" interaction itself is realized in societal contexts.

Therefore, the structure of interaction is always complex: at once realizing the addresser and the inner environment of the addressee. This simultaneity of inner and outer environments of agents is involving a kind of structural bifurcation and a mutual actions of *acceptance* and / or *rejection* of the involved agents based on complexity of their architectonics. That is, the addressee has to give space (einräumen) to the addresser to be addressed. To address and to accept to be addressed is a *mutual* action of at least two agents in a common co-created environment. Self-addressing would be a case of interactional reflectionality of an agent into itself.

Acceptance and rejection

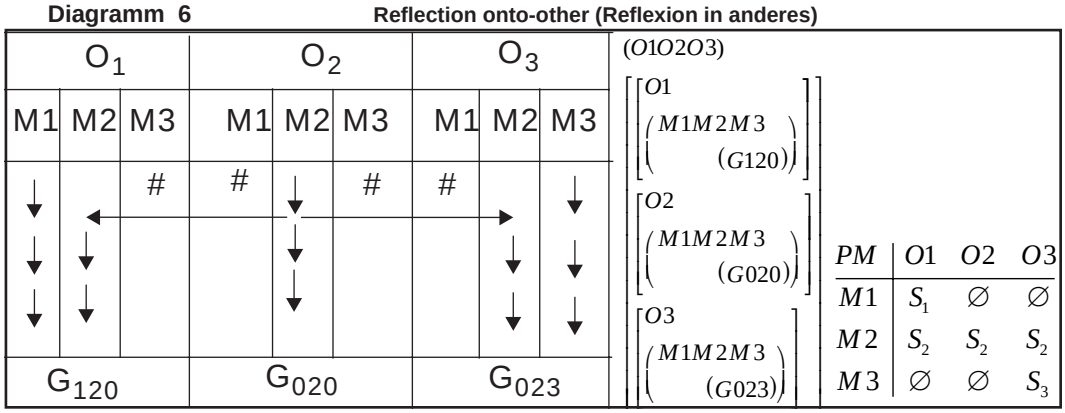
Interactivity therefore is a mutual action of *acceptance* and *rejection* between different agents. Only on the base of this interactional agreement information exchange can happen. Information, and information processing is, thus, secondary to interaction. Information thus belongs to cognition while interaction belongs to volition as the minimal contextures of subjectivity. This insight may not be new if we refer to phenomenological analysis of interaction and cognition well documented by Paul Dourish. The strict difference lies in the acceptance of the challenge to deal with it in formal polycontextural logical / mathematical and programming terms. An endeavour which is strictly rejected by the phenomenological approach.

Reductions of complexity

Also interactivity involves an architectonic representation of the neighbor systems to happen in a strict structural way it doesn't exclude the possibility of reduction to a less complex representation by an interacting agent. This is based on the dynamics between semantic and pragmatic aspects of interaction. What would enforce a structural place in a inner environment of an agent can be interpreted as a semantic interaction being well represented at an existing structural place shared by other interactional representations. In other word, a agent with 100 neighbors has not to represent for once and ever all 100 agents in his inner environment, simply because it is anyway not interacting in a structural way always and at once with his 100 neighbors. But there is no structural reason which would deny such a complex representation of neighbor systems if needed.

An example

Diagramm 6 **Reflection onto-other (Reflexion in anderes)**



In the example there is an addressing from O2 to O1 and O3 realizing positioning at once in O1 as O1M2 and in O3 as O3M2 while keeping O2M2 as the addresser of O1 and O2 persistent. The same simultaneous autonomy of agents holds for O1 and O3. The complexity of the agents O1 and O3 are giving space to the possibility of acceptance of the interaction.

In the diagram the mutual interaction of acceptance and rejection has to be interpreted, and is not visualized as such. That is, the action of O2M2 to O1M2 means, that O2 is rejecting the job and addressing it to O1 and O1 mutually accepts the job in offering computational space to O2 at O1M2.

Co-operation: Interactions as commands

The realization of an interaction, as described, can be understood as a co-operative command. An agent or processor O2 is commanding on the base of realized interactivity with agent O1 and O3, that is of realized architectonics, to proceed a task of O2M2 at O1M2 and the same with O3 to proceed at O3 a task of M2 at O3M2. On the base of realized interactivity the informational aspects of tasks can be considered.

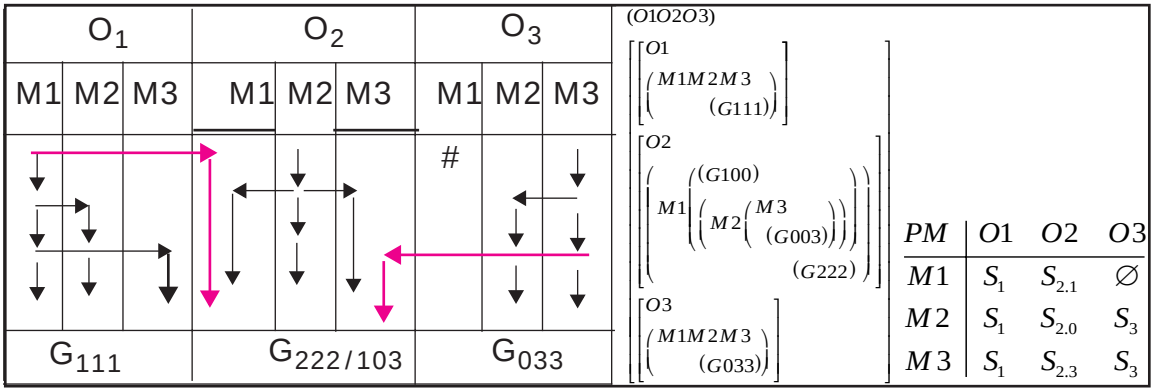
Cybernetic Ontology

This modelling of the situation is a further concretization of the concept of logical transjunctions introduced 1962 by Gotthard Gunther. In his paper *"Cybernetic ontology and transjunctional Operations"* the terminology of *acceptance* and *rejection* was central for the definition of the binary logical function *transjunction* in place-value systems (poly-contextural logics).

9 Interplay between Interactionality and Reflectionality

Mixing freely reflectional and interactional pattern are leading local iterations and recursions of the general scheme producing a fractalization of the general scheme.

In more systematic words, a full theory of computational subjectivity has to consider not only the aspects of reflectionality and interactivity as interpretations of cognition and volition in isolation but the full interplay of both together.



Pattern: [G111, G222/003/100, G033]

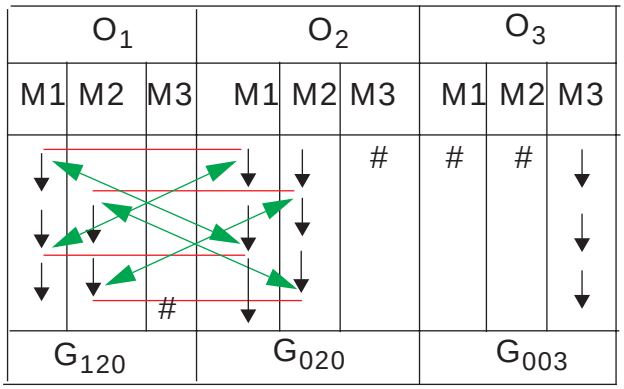
At the locus O2 we have a full introspection G222 and an interaction from the locus O3 into the locus O2 producing additionally to G222 (O2/M1M2M3/G222) the pattern:

O1: (M1M2M3)/G111

O2: ((M1/G100(M2(M3/G003))/G222)))

O3: (M1M2M3)/G033

9.1 Proemiality inside interactivity and reflectionality



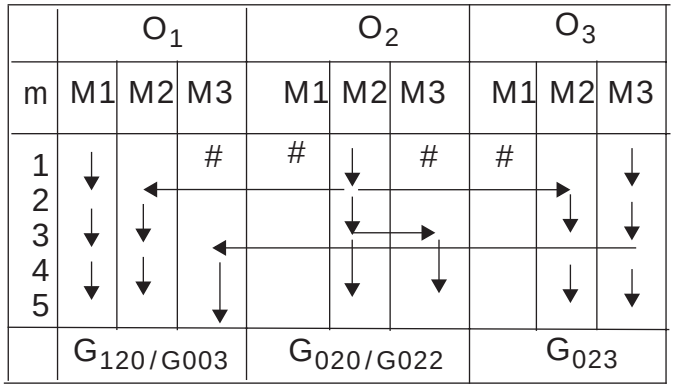
The exchange between operator and operand has to be described simultaneously from both positions. That is why we have to realize a double description, a double gesture of inscribing the proemiality of the constellation. To visualize this procedure we have to realize a double description of the diagram

The first diagrams are correct insofar as they describe the *structure* of proemiality. But at the same time they are abbreviations insofar as the *process* of reading them, that is to read them at once from both sides, is not inscribed. This process of reading has to be done by a reader. But we have to make it explicit and to visualize it. Therefore, even if it seems to be obvious, it has to be realized and not only be mentioned. The new diagram is focussing more the process of proemiality than on its general structure.

9.2 Superpositions of patterns

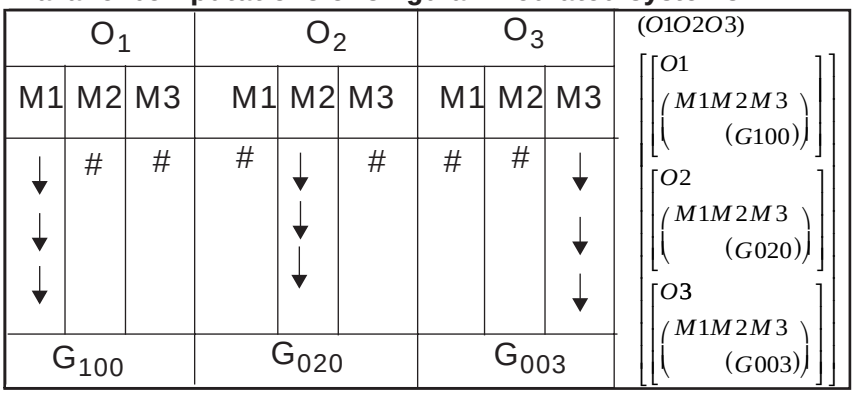
Superpositions are simply iterations of patterns and are additive to the matrices. The presentations until now are abstracting from the procedural steps. To introduce superpositions we have to add some organizational numbering to the diagrams representing the single steps at each locus. Each locus can have its own numbering, representing its own temporality of the procedural steps. There is no need for an universal clock for distributed systems. Time too, has to be disseminated and can be unified by negotiations between agents.

[S120, S020, S023] + [S003, S002, S023] = [S123, S022, S023}



10 Special constellations

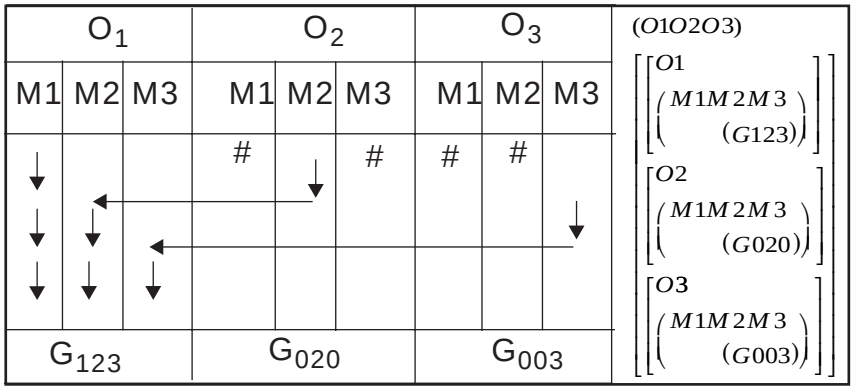
10.1 Parallel computations of singular mediated systems



In this case, the agents are not realizing their possibility of interacting and reflecting between each others. But nevertheless, they are not separated in abstract isolation because their distribution is mediated by construction. Which is also setting some limits in the range of their common rationality in restricting abstract combinations of operators to concrete realizable possibilities of mediated, but not interacting, combinations.

At each place, a full but not interactional/reflectional mediated programming and computation is happening. All sorts of strict mediated parallelisms can be realized in this setting.

10.2 Reductional interactions



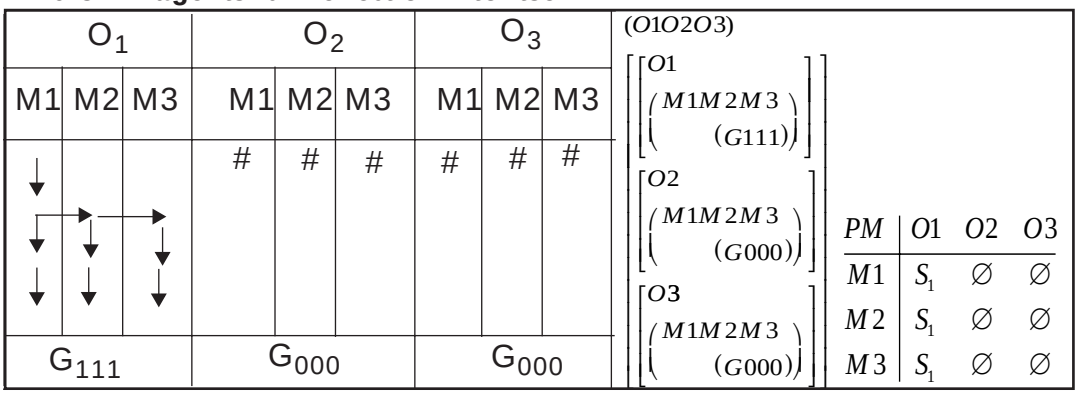
In this case the agents are reducing their polycontextural complexity to a single contexture but keeping their architectional design as 3-contextural.

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_2	$\varnothing$	$\varnothing$
$M3$	S_3	$\varnothing$	$\varnothing$

PM	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_2	S_2	$\varnothing$
$M3$	S_3	$\varnothing$	S_3

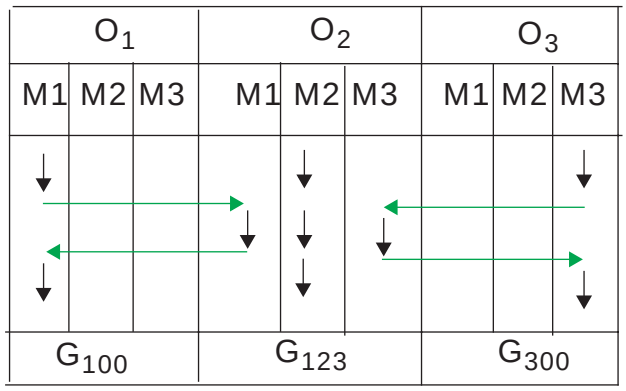
The first matrix shows the result of reduction to the pattern O1M123, the second keeps the origins of reduction. The reductions starts somewhere, therefore the pattern [S123, S020, S003] is not eliminating the sources of the reduction to the form [S123, S000, S000]. But this eliminative reduction pattern is correct for the *result* of the reduction as such. Both matrices make clear the difference to introspection produced by replication which is simply producing to the iteration of the single computational system (O1M1) to [S111, S000, S000].

10.3 An agents full reflection into-itself



At the place O1, the agent is reflecting into-itself the (active or) non-active but existing neighbor systems O2 and O3. This can be seen as a kind of reflectional *introspection* produced by the super-operator *replication* at 3 intra-contextural places. This structural limitation to 3 places [S111, S000, S000] is not excluding the endless iteration of reflection into-itself at the same place of reflection O1 in the stable constellation. That is, the pattern is fractal, the procedure iterative and bottom-less (co-algebraic).

10.4 Permutative Patterns



Permutations, produced by the super-operator *perm*, are behind theses "visits"to other system, mostly appearing in De Morgan like formulas of Boolean topics.

$$(O1O2O3)\left[\begin{array}{l}[O1\ (M1M2M3\ (G100)\ (G000)\ (G100))]\\ [O2\ (M1M2M3\ (G020)\ (G123)\ (G020))]\\ [O3\ (M1M2M3\ (G003)\ (G000)\ (G003))]\end{array}\right]$$

10.5 Stability of complexity and complication

```
interact (reflect(O1O2O3(M1M2M3))) = (O1O2O3(M1M2M3))
```

Stability: The constellation is closed under reflection and interaction.

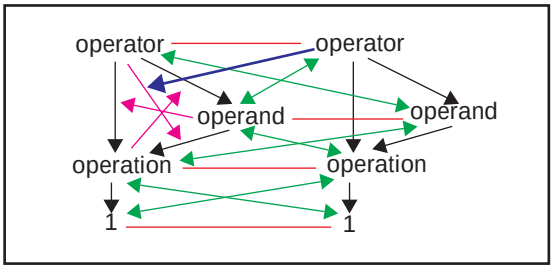
This explains why the reflectional domain of an agent is restricted to the number of its neighbor systems. If the scope of M would be enlarged without a general enlargement of the constellation it would have the function of introducing fictional and virtual neighbors. To enlarge the domains new functions of extensions would be needed. In all considered cases in this study, complexity as well as complication of the constellation is stable. The very interesting cases that interaction is augmenting complexity and reflectional complication of the constellation has to been studied later.

11 Intervention

Interaction as reflection: reflectional interactivity (intervention)

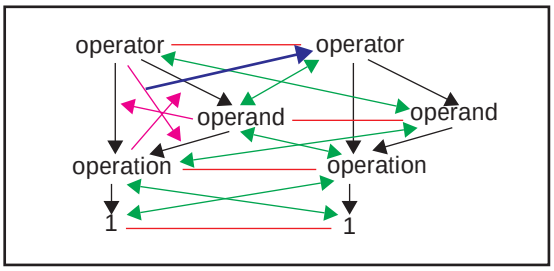
Reflectional interactivity can be understood as an interaction unto the reflectional patterns of a neighbor agent or into the acting agent itself, therefore it can be called intervention and self-intervention.

Interventions are anticipating the behavior of an agent and try to influence it and to change its plans and motivations maybe to avoid conflicting situations.



Intervention is re-programming the reflectional system of the neighbor system and not the system itself. The self-image of the neighbor system is re-programmed and not the system itself as it appears in an interactional context to the interacting agent and also not as the reflectional image of the neighbor in the internal environment of the agent. This is a further specification of subjectivity in the I/Thou-relation of togetherness or the proemiality of (cognition, volition, I, Thou) in the sense of Gunther.

Interventions may be realized in two directions of reflection and interaction, reflectional interactions.



Societal Activity

$Interaction: \quad (O, M) \rightarrow (O, M)\ with\ Oi \rightarrow Oj$

$Reflection: \quad (O, M) \rightarrow (O, M)\ with\ Mi \rightarrow Mj$

$Intervention: \quad (O, M, I) \rightarrow (O, M, I)\ with\ Ii \rightarrow Ij$

$Anticipation: \quad (O, M, A) \rightarrow (O, M, A)\ with\ Ai \rightarrow Aj$

$SozAction: \quad (O, M, I, A) \rightarrow (O, M, I, A)$

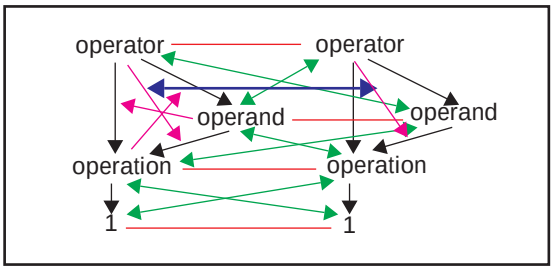
11.1 Intervention between reflectional systems

From; "A Formal Method of Investigating Reflective Processes", V.A. Lefebvre, General Systems, Vol. XVII, 1972, pp.181-198

"We denote the conflicting parties (individuals) by X, Y, and Z. To make a decision, X must construct a model of the situation (for instance, to represent in a particular way the arena of interaction, together with the forces at hand). Y also constructs a model that his opponent X also has some model of the situation. Z, in turn, may be aware that the inner worlds of X and Y are constructed in just this manner.
Succes in conflict is largely determined by how adeaquately the opponents simulate each other's inner worlds."

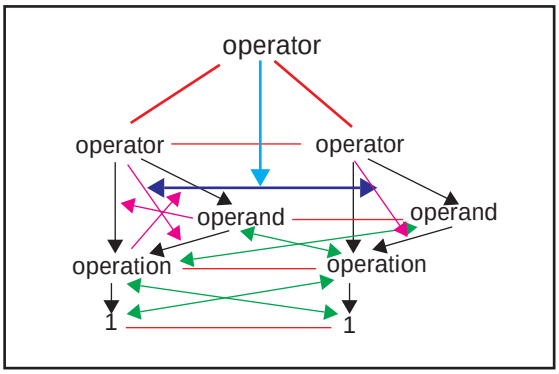
12 Interlocution (Anticipation)

Reflection as interaction: interactional reflectionality (interlocution, anticipation)



Interactional reflectional can been seen as a one directional or a mutual interaction between two reflectional activities. Plans, motivations and strategies are directly involved with the aim to interact or change each others intentions and self-interpretations.

Inner and outer describtion of the arena of reflections and interactions.



D. Mapping ARS onto the PCL-Matrix

13 Abstract of ARS

13.1 ARS (by Georg Loczewski)

- *Abstraction*:: to give something a name,
- + *Reference*:: to reference an abstraction by name,
- + *Synthesis*:: to combine one abstraction with other abstractions to create something new.

The constitutive principles of A++ are those, that make A++ to what it is. These principles are essentially the nucleus of the language, everything else can be derived from them. ARS, as introduced above in ars provides three of these principles and `lexical scope' is the fourth.

In A++ ARS is *universal*, the principles can be applied anywhere at any time because they make up the language.

Commenting the definition

- Like the introduction to the Lambda Calculus the definition of A++ deals with `lambda expressions' and defines three different types: `abstractions', `references' and `syntheses'. This corresponds to the `lambda abstractions', the `variables' and the `applications' in the Lambda Calculus. The few differences in the A++ - approach are the following:
- The syntax of A++ is different. It is borrowed from Scheme and so simple that it can be described in a few words:

A lambda expression may either be a symbol or a construct *with* parenthesis*. A construct with parenthesis represents a lambda abstraction if the first thing following the opening parenthesis is a symbol '*lambda*' or '*define*', otherwise it must be an application.*

This is all there is to the syntax! It can't be simpler.

- Abstractions may be given a name explicitely, matching the general human understanding of `abstraction' as `to give something a name'.
- Abstractions may contain more than one lambda expression in the body to be evaluated.
- Applications may contain more than two lambda abstractions including several arguments passed to the operator.
- The rules for the conversion of lambda expressions defined in the Lambda Calculus are valid in A++ as well. Due to `lazy evaluation' in A++ lambda expressions can be treated the same way as in the Lambda Calculus.

Lexical Scope

*`Lexical Scope'* defines the access to variables within functions. Variables within functions are either `lambda-bound' or `free'. The `lambda-bound' variables refer to arguments passed to the function. The so-called `free' variables must have been defined in the inherited environment of the function. In a language with `Lexical Scope' a lambda abstraction inherits all variables from those abstractions in which it is defined.

`Lexical Scope' in A++ is coupled with indefinite extent in contrast to Pascal where 'Lexical Scope' is coupled with `limited extent'.

ConTeXtures = DISS (ARS)

DISS = Distribution + Mediation

ConTeXtures =
Computation +
Reflection +
Interaction +
Intervention +
Anticipation

Definition of A++ in EBNF-Notation

```
<expression>  ::= <abstraction>  |
                  <reference>    |
                  <synthesis>
<abstraction>  ::= '(' define <variable> <expression> ')'  |
                                '( lambda (' {<variable>}')'  |
                                <expression> { <expressions> }')'
<reference>   ::= <variable>
<synthesis>    :: = '(' <expression>  { <expression> }')'
<variable>    ::= <symbol>
```

- | vertical bar stands for 'or'
- [...] brackets mark optional expressions
- { ... } curly braces mark repeated expressions: 0 or n-times
- ' ... ' single quotes mark literal text
- (...) braces are used for grouping

- < ... > angle brackets mark terms

The generalizations of the Lambda Calculus are threefold:

- An abstraction can be assigned a name, and such an abstraction with its name definition may appear anywhere in a program.
- The body of a lambda abstraction may consist of more than one lambda expression.
- A synthesis may combine more than one lambda expression.

http://www.lambda-bound.com/
http://www.aplusplus.net/bookonl/node28.html to /node36.html

13.2 Syntax of the Lambda Calculus

Because of my extensive use of the operator/operand terminology for the distribution and mediation of ARS, a citation of Burge may be helpful:

"An expression may occur in three positions as a component of a larger expression:
1. in the operator position,
2. in the operand position,
3. as the body of another lambda expression.

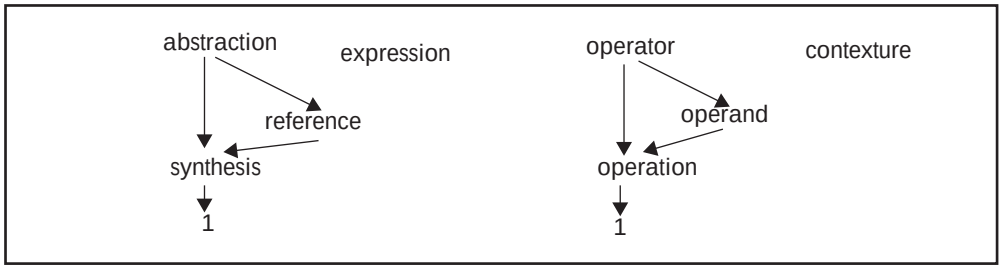
The lambda expression is the second basic method of assembling a new expression. In their most austere form the expression under consideration may be characterized as follows.

An expression is
    either *simple* and is an identifier
    or a l*ambda expression*
        and has a *bound variable* which is an identifier
        and a body which is an expression,
    or it is *composite*
        and has an *operator* and an *operand*, both of which are expressions.

A rule is needed for recognizing when the body of a lambda expression ends. The rule is that the body extends as far as it can until it is terminated by a closing bracket, comma, or the end of the whole expression. It follows that parenthesis are only needed to enclose the body if it is a list although they may be used if this improves readability."
W.H. Burge, Recursive Programming Techniques, 1975, p.9

13.3 Operational interpretation of ARS

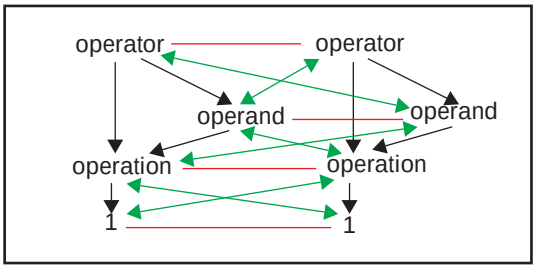
ARS-syntax	Operator-terminology
<expression> ::= <abstraction> | <reference> | <synthesis>	linguistic ARS-**contexture** ::= <operator> | <operand> | <operation>
<abstraction> ::= '(' define <variable> <expression> ')' | '(lambda (' {<variable>}')' | <expression> { <expressions> }')'	**<operator>** ::= operator of operator (operator as operator) operator of operand (operator as operand) operator of operation (operator as operation)
<reference> ::= <variable> <variable> ::= <symbol>	**<operand>** ::= programming operand <operand> ::= lingustic operand
<synthesis> :: = '(' <expression> { <expression> }')'	**<operation>** ::= operator (operation)



Comments and contrasts

It is more than obvious that the basic structure of ARS is in all senses of its introduction hierarchic, and there is no justification and formalization of its hierarchy by the calculus itself. Also the conceptuality of ARS is deliberating the classical use of the lambda abstraction internally to a profound self-referentiality of the naming process it is stacked strictly in its logocentric closure. In contrast to this hierarchic structure, ConTeXtures are based on the *intuition* of proemiality which allows a heterarchic distribution and mediation of different ARS systems as wholes.This is realized by the acceptance of the full interplay of its terms, operator, operand, operation and position.

The strict *linearity* and *hierarchy* of the operationality of the Lambda Calculus is thus replaced by the *heterarchy* and *tabularity* of ConTeXtures structured by the interplay of the proemial relationship between operator, operand, operation and positions.



The diagram gives only the minimal structure of this interplay. Polycontexturality is not starting with one, also not with 2 distributed systems, but with 3. The third system is mediating the 2 mediated systems. Its balanced structure is introduced with 4 basic systems involving 6 contextures.

It should be emphasized that the two representations, the ARS and the ConTeXtures, allows to make the difference between "ontological" entities and reflectional relations. That is, in ARS: abstraction = abstraction. ConTeXtures are guided by the reflectional as-abstraction: X as Y is Z.

13.4 Mapping the principles of ARS onto polycontextural architectonics

To implement ConTeXtures (SAMBA'S) the programming concept A++ (ARS) has to be mapped onto the general pattern of polycontexturality.

The operator "*" stand for the *mediator* of sambas, the number "3" tells the degree of the distribution by the *distributor* of samba.

The mediator is defined by the proemial relationship of (coinc, exch, ord).

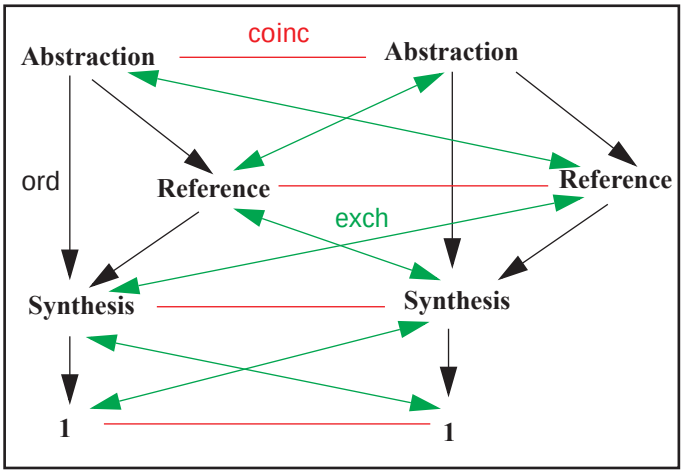
The operator "coinc" is representing the binary relation of coincidence, in the sense of the sameness of two operands or operators.

The operator "ord" is representing the binary relation of order, in the sense of the asymmetry of an ordered pair of operator and operand.

The operator "exch" is representing the binary relation of symmetrical exchange, in the sense of a symmetric difference between operator and operand.

13.4.1 Architectonic proemiality of SAMBA'S

14. Proemiality of (ARS, 2) = samba (Abstraction, Reference, Synthesis, 2) with:

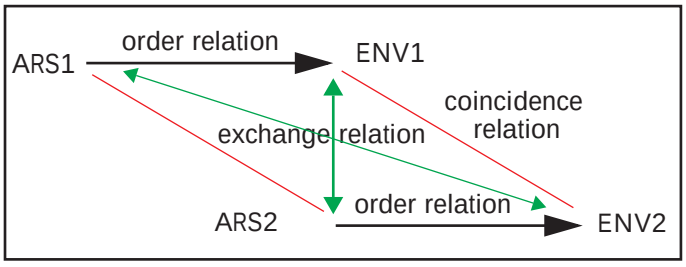


abstr1 conc abstr2
refer1 coinc refer2
synth1coinc synth2
abstr1 exch refer2
abstr1 exch synth2
refer1 exch abstr2
refer1 exch synth2
synth1 exch abstr1
synth1 exch refer2

A systemic distribution of ARS is considering the difference of system and its environment involved in the definiton of ARS (cf. SUSHI'S LOGICS).

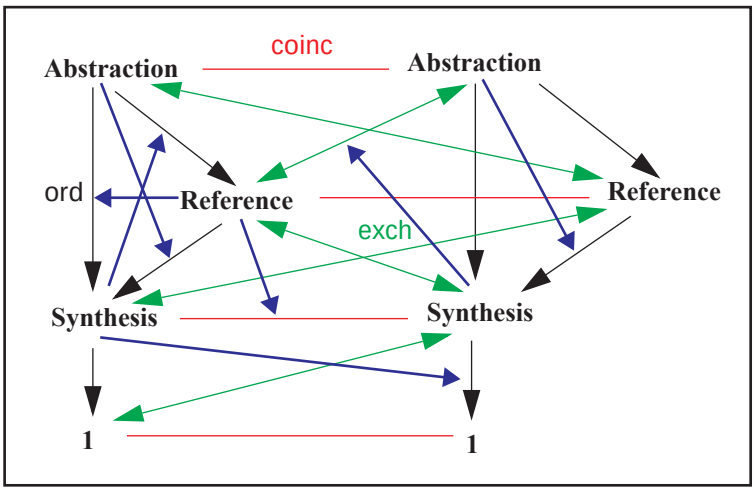
15. Proemiality of ARS and its Environment: samba (ARS, ENV, 2):

ARSi ord ENVi,
ARS1 coinc ARS2
ARS1 exch ENV2
ENV1 coinc ENV2
ARS2 exch ENV1.



"The formation of an abstraction in A++ is not an absolute event, detached of all. An abstraction always takes place in a certain context, which belongs thus substantially to the formed abstraction. Lambda abstraction is connected at the time of its production with their context or their environment. The result of this encapsulation is called 'Closure'. " Loczewski (translation, r.k.)

The new formal system is not dealing with the excluded nonsensical syntactical combinations of the former system (env), but with the otherness of this system which has two meanings: metaphorically, the shadow of one system and the simultaneous brightness of the other.



This diagram gives an overview of the main ways of thematizing the architectonics of disseminated ARS systems. From proemial dissemination as *interaction* to *reflectionality* inside a system, to *interventions* and *anticipations* of relations between ARS systems.

13.5 poly-ARS: Mapping ARS into the polycontextural Matrix

The proposed matrix has two dimensions interpreted as interactional and reflectional dimensions. ARS is distributed over these two dimensions. All laws of and features of ARS is repeated and conserved by the distributed ARSs. Mediation of distributed ARS systems is not conflicting the internal laws and features of the distributed systems because these distributions, reflectional or interactional, are disjunctive separated from each other. Therefore, meta-logical topics for each ARS, like consistency, computability, correctness, completeness, efficiency, etc. are inherited. The distribution is not destructive to the internal structure of ARS systems.

PM	$O1$	$O2$	$O3$
$M1$	$\varnothing$	$\varnothing$	$\varnothing$
$M2$	$\varnothing$	$\varnothing$	$\varnothing$
$M3$	$\varnothing$	$\varnothing$	$\varnothing$

===>

PM	$O1$	$O2$	$O3$
$M1$	ARS	ARS	ARS
$M2$	ARS	ARS	ARS
$M3$	ARS	ARS	ARS

Also reflectional and interactional systems are generally generated from the point of view of basic ARS systems, ARSi,j, i=j, *diagonal* systems, by means of the superoperator, from an architectonic viewpoint, each position OiMj is containing, is giving space principally to a full ARS system. What will be restricted only are the possible combinations of mediations of different ARS systems violating conditions of mediation.

Each system can interact with its neighbor systems and can reflect / mirror in itself these neighbor systems. Thus, to the 3 diagonal systems, 6 further reflectional and interactional systems are involved. That is, each system can interact with the neighbor systems of its outer environment and can have a reflectional representation of them in itself, in its inner environment.

14 Mapping Interactivity

14.1 Interactive complexity of poly-ARS

Because of the genuine interactive structure of distributed formal systems, ARS in a complexion is not fully defined by its isolated structure. ARS-systems in poly-A++ don't come in isolation, they are mediated. To focus on a single or a group of ARS-systems has to take in consideration their interactive structure.

```
Mediation
samba (ARS, 3) = ARS1 * ARS2 * ARS3

ARS(m) : ARS(m) ---> ARS(m)
locus 1 : ARS1.1, ARS1.2, ARS1.3       for m=3
locus 2 : ARS2.1, ARS2.2, ARS2.3
locus 3 : ARS3.1, ARS3.2, ARS3.3
```

Acceptance and rejection are interactive topics, procedures, operators which are mirrored in the neighbor systems on their internal environments. This has to be reflected in the architectonics of the complex system.

Interactivity is transforming or keeping the structure of the complexion (group of agents) as whole and is not (yet) considering the actions of single agents in a group.

Also we are using often a linear notation for poly-ARS it is obvious that this can be only a technical abbreviation of the genuinely tabular structure of poly-contextural systems.

14.1.1 Superoperators of SAMBA'S

samba (sub-op, ARS, n)

Super-operators **sup-op** :={**id, perm, red, bif, trans, repl**}

```
samba [(id, perm, red, bif, transition, replication), ARS, n]
```

16. Identity

```
samba (id, ARSi) = ARSi
```

17. Transition

```
samba (trans, ARSi) = ARSi+1
```

18. Permutation

```
samba (perm, ARSi, ARSj) = (ARSj, ARSi)

perm1 : ARS123 ---> ARS132
perm2 : ARS123 ---> ARS321
```

19. Reduction

```
samba (red, ARSi, ARSj) = (ARSi, ARSi)

red1 : ARS123 ---> ARS133
red2 : ARS123 ---> ARS112
```

20. Bifurcation

```
samba (bif, ARS, ...ARSi, ...ARS) =
            ( ARS, ...(ARSi1...ARSin), ...ARS))

bif1 : ARS123 ---> ARS1, ARS2.1.3, ARS3
bif2 : ARS123 ---> ARS1.2.3, ARS2, ARS3
bif3 : ARS123 ---> ARS1, ARS2, ARS3.1.2
```

21. Replication

```
samba (repl, ARS, ...ARSi, ...ARS) =
              ( ARS, ...(ARSi1...ARSin), ...ARS))

repl1.2 : ARS123 ---> ARS1, ARS2.1, ARS3
repl1.3 : ARS123 ---> ARS1, ARS2, ARS3.1

repl2.1 : ARS123 ---> ARS1.2, ARS2, ARS3
repl2.3 : ARS123 ---> ARS1, ARS2, ARS3.2

repl3.1 : ARS123 ---> ARS1.3, ARS2, ARS3
repl3.2 : ARS123 ---> ARS1, ARS2.3, ARS3
```

22. Reflection

```
samba (refl, ARS, ...ARSi, ...ARS) =
              ( ARS, ...(ARSi1...ARSin), ...ARS))
```

All realizable combinations of the super-operators have to be introduced.

```
Super-operator of interactivity and reflectionality:
sup-op = {id, perm, red, bif, repl}

{id, id,}
{id, perm}
{id, red}
{id, bif}
{id, repl}
{id, repl, perm}
{id, repl, bif}
{id, repl, perm, bif},   etc.
```

Scheme for patterns: [(x, y, z)]

14.1.2 General pattern for the (id, id, id)-modus of interactivity

$$
\begin{array}{l}
samba^{(3)}(id,id,id) - horizon \\
\left[\begin{array}{l}
thematize\ (topics^{(3)}) \\
\left[\begin{array}{lll}
\left[\begin{array}{l} identify\ contexture_1 \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array}\right) \end{array}\right) \end{array}\right] & \varnothing & \varnothing \\
\varnothing & \left[\begin{array}{l} identify\ contexture_2 \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array}\right) \end{array}\right) \end{array}\right] & \varnothing \\
\varnothing & & \left[\begin{array}{l} identify\ contexture_3 \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array}\right) \end{array}\right) \end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}
$$

Template: [S100, S020, S003],
Pattern: [id, id, id]

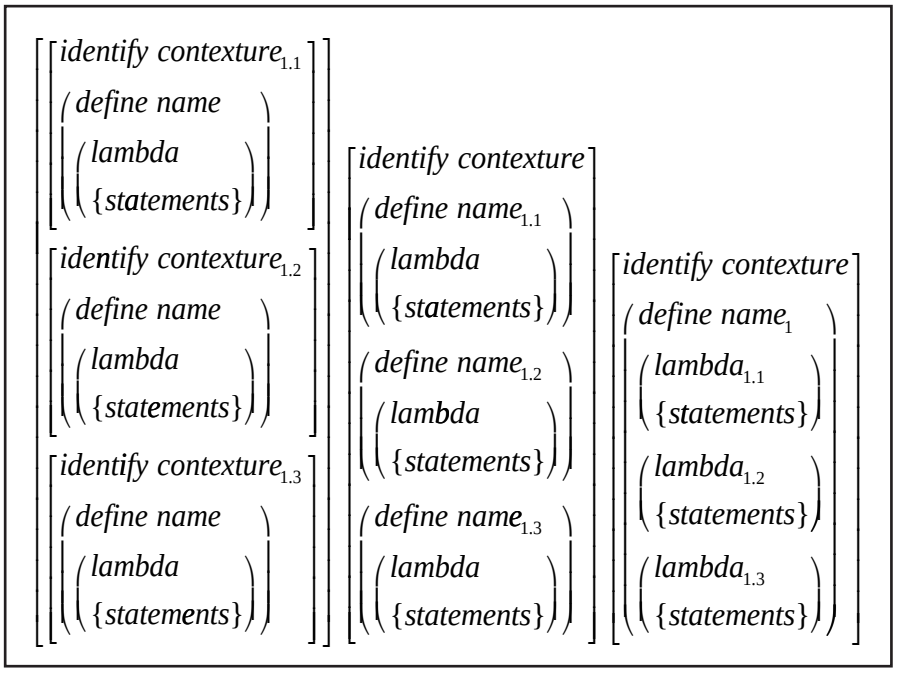
14.2 Tectonics of Interaction

Different modi of reflection produced by the operator of replication can be defined with the help of the distinction contextures / heads and body. Thus, tectonic patterns of interactivity are defined by the combination of the contexture / head / body distributions. This is continued by the reflectional patterns concerning tectonics.

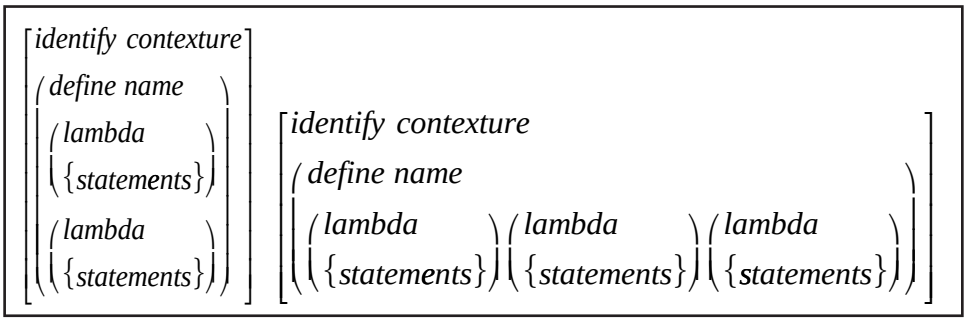
23. *Metamorphic* interaction: Replication of contextures at a single locus including different head / body / statements.

24. *Alterational* interaction: Replication of heads into itself including bodies under single contextures.

25. *Replicational* interaction: Replication of bodies into itself including statements under single heads.



26. *Iterations* of bodies in interactional situations which are ruled by a head under a single contexture are not interactions but *superpositions*. It is only a question of notation to write them in a horizontal or in a vertical way.



15 Mapping Reflectionality

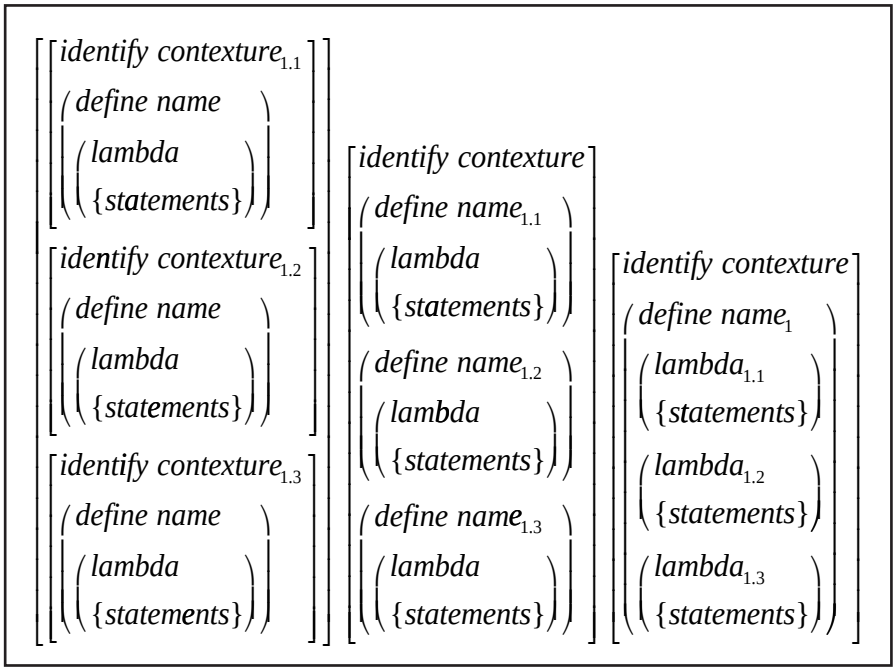
15.1 Tectonics of Reflection

Differend modi of reflection produced by the operator of replication can be defined with the help of the distinction contextures / heads and body.

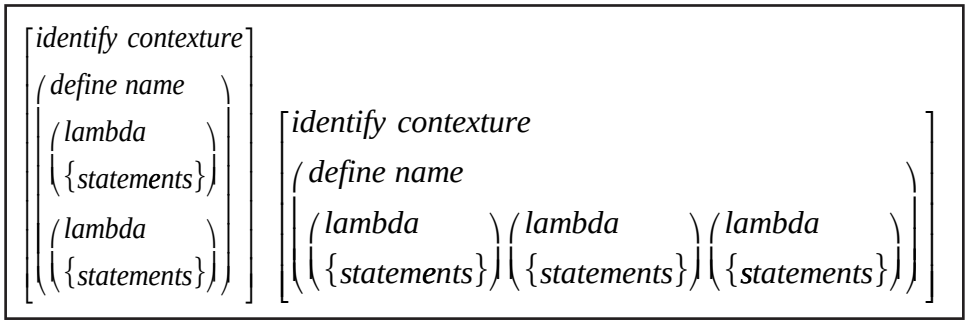
27. *Metamorphic* reflection: Replication of contextures at a single locus including different head / body / statements.

28. *Alterational* reflection: Replication of heads into itself including bodies under single contextures.

29. *Replicational* reflection: Replication of bodies into itself including statements under single heads.



30. *Iterations* of bodies in reflectional situations which are ruled by a head under a single contexture are not reflections but *superpositions*. It is only a question of notation to write them in a horizontal or in a vertical way.



15.2 Reflectional super-operators / super-operator in reflectional contexts

Traditional terms: refection, introspection, meditation, self-reflection, self-modification.

Reflection and interaction are complementary features of ConTeXtures, this is mirrored in the applications of the super-operators.

Also reflection is ruled and build up by the architectonic operator *repl* different super-operators, manipulating the reflectional order, can be introduced. The order of reflection can be permuted by *perm*, reduced by *red* and split by *bif.* The same operators as in the case of interactivity can be applied. That is, these operators have to be differently implemented according to their domain of reflectionality in contrast to interactivity. Super-operators are changing the indices of their objects and are more or less neutral to the distinction of head / body as for the classification of specific reflectional patterns.

Simple reflection into-itself of a contexture by replicating the body of a definition.

$$
\begin{array}{l}
samba^{(3)}(repl, \varnothing, id) - horizon \\
\left[\begin{array}{l}
thematize\ (topics^{(3)}) \\
\left[\begin{array}{ccc}
\left[\begin{array}{l}
identify\ contextures \\
\left[\begin{array}{l}
define\ name \\
\left[\begin{array}{c}
\begin{pmatrix} lambda_{1.1} \\ \{statements\} \end{pmatrix} \\
\begin{pmatrix} lambda_{1,2} \\ \{statements\} \end{pmatrix} \\
\varnothing
\end{array}\right]
\end{array}\right]
\end{array}\right] & \begin{array}{c} \varnothing \\ \varnothing \\ \varnothing \end{array} & \begin{array}{c} \varnothing \\ \varnothing \\ \left[\begin{array}{l} identify\ contexture_{3.3} \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{array}\right] \end{array}
\end{array}\right]
\end{array}\right]
\end{array}
$$

Pattern: [S_{110}, S_{000}, S_{003}],

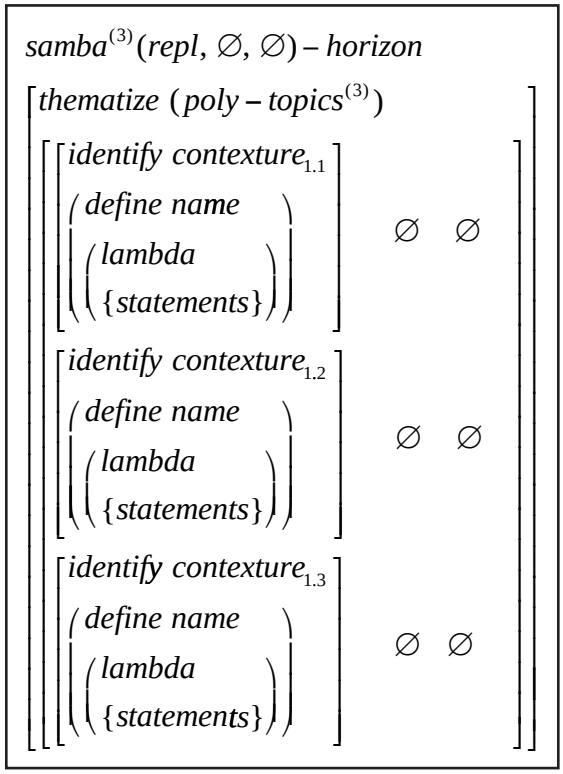
Template: [(repl, repl, empt), (empt, empt, empt), (empt, empt, id)]

Without metamorphosis, reflection is stable in respect to its topics. Its topics don't change under reflection. But there are also good reasons to study metamorphic reflections which are changing the definition of the topic of the reflected object. This happens in creative reflections when an object turns out to be of another topic then believed at first. In this case the general pattern has to be transformed.

Reflectionality is connected with the ability of the operators samba and contexture to call a contexture. It seems that this is realized by building and calling the full closure of a contexture.

15.2.1 Replicational and metamorphotic reflection.

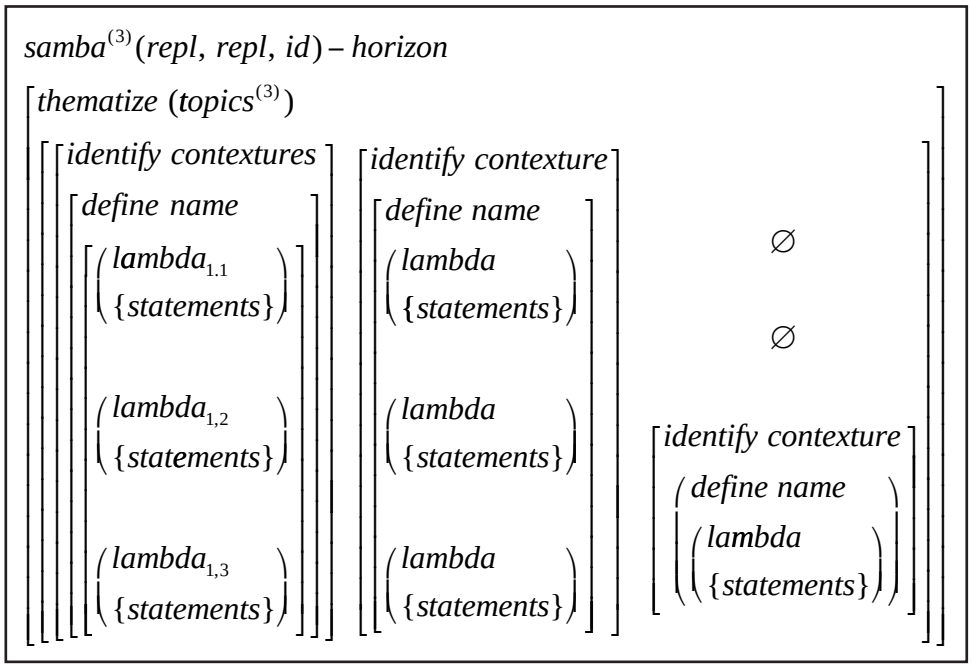
A pattern for metamorphic reflexion



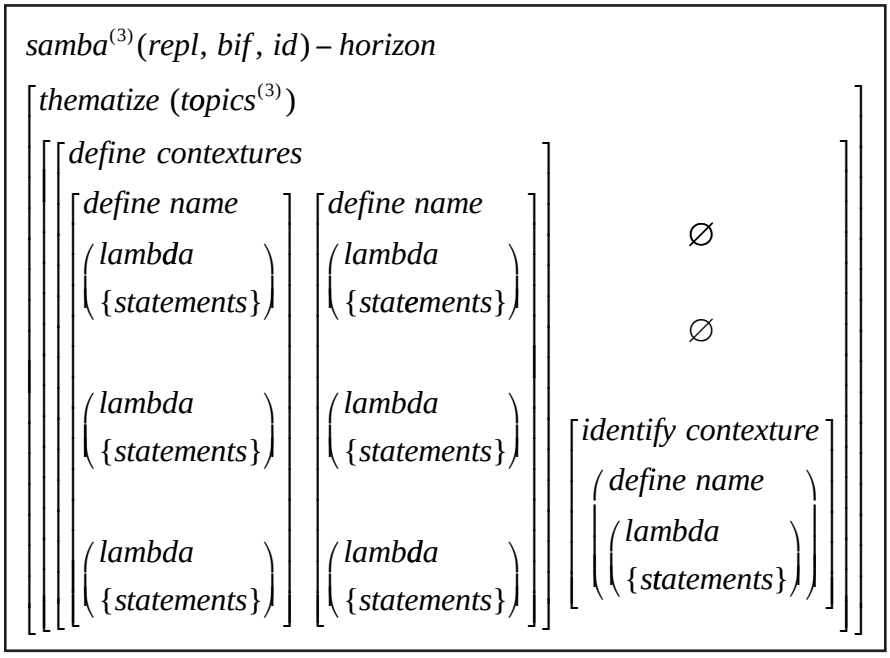
Pattern: [S_{111}, S_{000}, S_{000}]

The succession of reflexion and the successive transformation of the topic of the object seems to be not restricted by reflectional mediation rules.

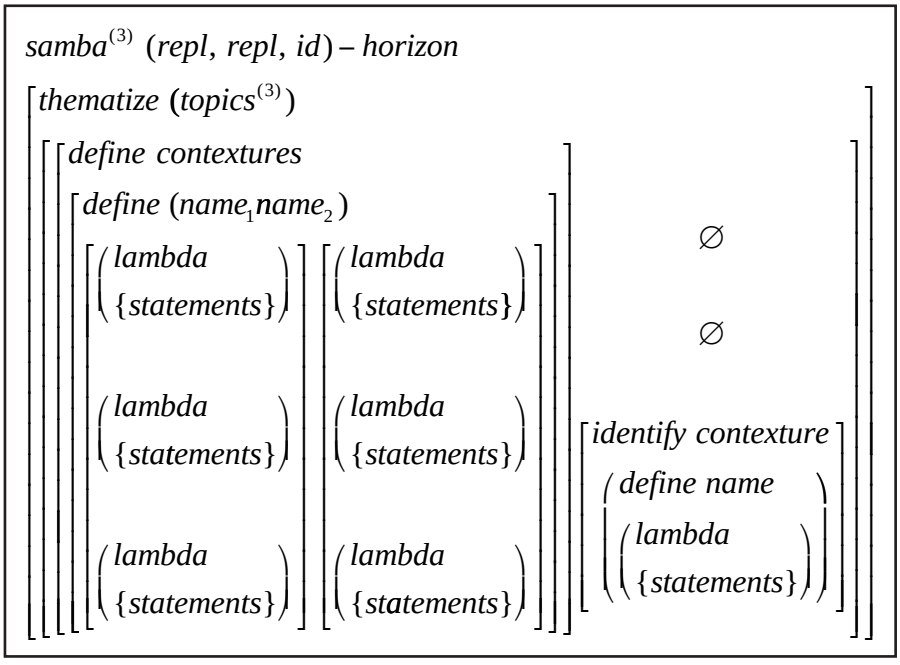
In contrast to this, it has to be analyzed which combinations of topics can be mediated under poly-topic distribution over different non-reflectional contextures.

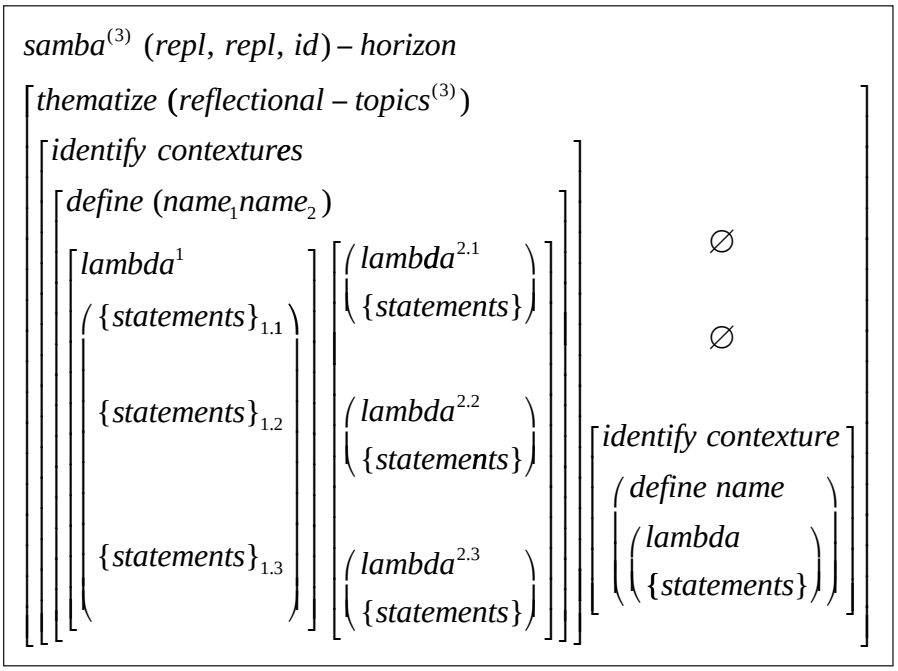

samba(3)(repl, repl, id) – horizon
thematize (topics(3))
identify contextures
define name
(lambda1.1 {statements})
(lambda1,2 {statements})
(lambda1,3 {statements})
identify contexture
define name
(lambda {statements})
(lambda {statements})
(lambda {statements})
∅
∅
identify contexture
(define name (lambda {statements}))

15.3 Combinations of interactivity and reflectionality




```
Pattern: [S111, S222, S003]

[((contextures1, contexture2),
(head1, (body1.1, body1.2, body1.3)),
(head2, (body21, body2.2, body2.3))),
(contexture3, head3, body3)]
```




samba(3) (repl, repl, id) – horizon
thematize (reflectional – topics(3))
identify contextures
define (name1name2)
lambda1
{statements}1.1
{statements}1.2
{statements}1.3
lambda2.1
{statements}
lambda2.2
{statements}
lambda2.3
{statements}
∅
∅
identify contexture
define name
lambda
{statements}

E. Mapping General Topics

Topics are specifications of the general objects (elements, terms) of ConTeXtures. General polycontextural objects are called complex objects, *c-obs*. Intra-contextural objects are the well known *obs* of Curry. Topics are specified into the data types or sorts of Boolean, Numeric, List, Relational, Class, etc. and reflectional / interactional realizations.

31. Boolean
32. numerical
33. list
34. class
35. interactional
36. reflectional (paradoxial, ambiguous)

16 Objectional proemiality of SAMBA'S: From obs to c-obs

The general scheme is: **samba ((ARS), n)**

samba (ARS), 1) = (ARS)

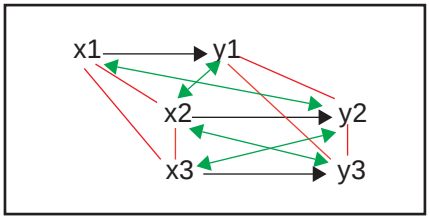
samba ((ARS), 3) = (ARS* ARS* ARS)

```
samba((lambda x y), 3)
        (lambda x y)
             (lambda x y)
                  (lambda x y)
```

samba (samba, 3, (ord, coinc, exch), x, y)

coinc (x y)
x1 coinc x2
x1 coinc x3
x2 coinc x3
y1 coinc y2
y1 coinc y3
y2 coinc y3

ord (x, y)
xi ord yi, i=1,2,3



exch (x y)
x1 exch y2
x1 exch y3
y1 exch x2
y1 exch x3
x2 exch y3
x3 exch y2

$x^{(3)}$ == (x^1 coinc x^2 coinc x^3)

$y^{(3)}$ == (y^1 coinc y^2 coinc y^3)

$x^{(3)}$ exch $y^{(3)}$ $ $x^{(3)}$ ord $y^{(3)}$

ARS as a special case of ConTeXtures

ARS = samba ((ord) x y, 1)

samba ((ord, coinc, exch), x,y,1) = samba (ord, x,y, 1) =

samba ((ARS), 1) = ARS

The basic function or structure "ord" is behind the basic ARS function "sel".

But, there are other possibilities usually not recognized by formal approaches:

analogism = samba (ord, coinc, exch), 1) ==> samba (coinc, 1)

polarism = samba (ord, coinc, exch), 1) ==> samba (exch, 1)

And all combinations of incomplete chiasms. (cf. Table)

As in Lambda Calculus and Combinatory Logic the variables x and y stand for highly abstract objects, their reference is called "*obs*" by Curry. These obs are neutral to characterization as Booleans, Numbers, Elements, etc. Obs as objects of a contexture are strictly identical with themselves, but in a polycontextural environment they are involved in an interplay of sameness with their neighbor objects, too. In this interplay they are highly ambigue objects, called complex objects, *c-obs*. At this point, I have to refer to the introductory chapters to Polycontextural Combinatory Logics.

Abstraction as *"giving something a name"* is neutral to all kinds of identive objects. Thematization as *"identifying something as a (con)texture among other contextures"* is neutral to all kind of complex objects, c-ob, "refered" or "evoked" by the operation *thematize*. Linguistically, thematizing is not referring but evoking. If the *evocation* (lit. Webster, 'imaginative recreation') is stable it can turn to the mode of reference. Evocation is a hermeneuticat process, it is not identifying but interpreting textures.

A programming (con)texture is not a single programming language, or a prototype of a language, but a complex cluster of mediated different programming languages involved in mutual interactivity.

Identity versus sameness

"If y *is an* ob*, then* y = y*."* (Curry, 1968)

If $y^{(m)}$ is an $c\text{-}ob^{(m)}$, with (for m=3)

Locus1: If y *is an* ob*, then* y = y

Locus2: If y *is an* ob*, then* y = y

Locus3: If y *is an* ob*, then* y = y

then $y^{(m)}$ *same* $y^{(m)}$

and identity for yi, yj, i=j: yi = yi, but not-identity for i /=j: non(yi = yj).

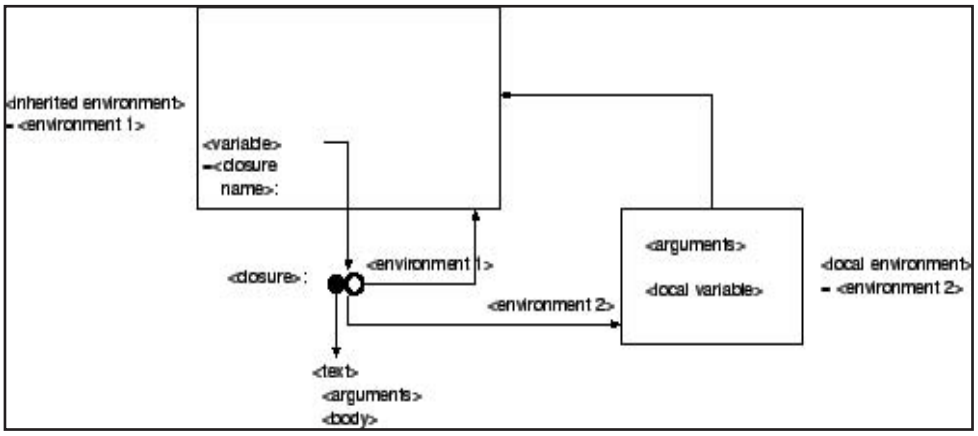
yi same as yj , but not identical.

$samba^{(m)}$ (... ($samba^{(n)}$ ...)) —> $samba^{(m+n)}$ (...)

17 Mapping the Closure Pattern

17.1 Closure pattern in A++

The most fundamental of the general programming patterns derived from ARS is the closure pattern. A closure is an *encapsulation of a lambda abstraction with its total environment.* This environment consists of *all the names* that this lambda abstraction has access to. Access to names in a lambda abstraction is controlled by the so called `lexical scope'.
Lexical scope can also be described as the context of a lambda abstraction in the program text.



A closure is a first class object, which means that it can be treated like any data item:

* it can be stored in memory,
* it can be passed as an argument to a function and
* it may be returned as a value from a function.

The slight difference between a closure and an object in OOP is the following:
* A *closure* is essentially a function that can be called, but may have all kinds of data and procedures encapsulated in it.
* An *object* is essentially a data item with all kinds of data and procedures encapsulated in it. One of its procedures (normally called methods) may be `apply', allowing to apply the object to a set of arguments, which essentially is the same as calling a function.

http://www.aplusplus.net/bookonl/node50.html

17.2 Distributed and generalized Closure patterns in poly-A++

Because of the introduction of the operator of thematization as a generalization of the mono-contextural operator of abstraction in ARS, thematization is demanding in the same way for a generalization of the concepts of *closure* and *lexical scope*. Additionally to this generalization, aspects of interactivity and reflexionality between different closures have to be studied.

Intra-contextural closures are defined as closures in A++. The difference lies in there multitude in contrast to the singularity of the closure definition introduced by Loczewski.

Multi-closures are introduced by the new concept of (poly-contextural) thematization in generalizing the concept of (intra-contextural) abstraction.

Thematizations are producing complexions of closures.

The metaphor is therefore not a clam (c-lam) but a *rhizomatic* population of same and different clams.

Contextures as closures / Closures as contextures

Contextural Heads

Additional to the distribution of closures to a complexions of closures a new generalization of the closure and lexical scope has to be established.

The new environment is now not data- or object-based but operator- and super-operator based. That is, the new environment is build by the operators of the structural heads and their interactivity / reflectionality.

Polycontextural closures are organizing their contextural closures heterarchically.

samba (thematize ...) is defining the structural head of a complexion.

...(define (lambda ...)) is defining the intra-contextural body of a singular contexture.

The generalized concept of closure can be understood as a step to a structural definition of the autonomy of interacting complex systems.

18 Basic patterns and topics in ConTeXtures

What has to be introduced, first in imitating A++, are basic intra-contextural topics like *numbers*, *Booleans*, *Lists*, *Objects*, etc. and new trans-contextural patterns of *interactivity* and *reflectionality* like patterns of introspectional deepness.

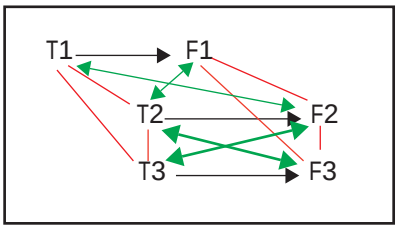
Patterns are the structures of interactivity and reflectionality.

Topics are specifications of the general objects (elements, terms) of ConTeXtures. General polycontextural objects are called c-objects. Intra-contextural objects are the well known obs of Curry. Topics are specified into the branches of Boolean, Numeric, List, Relational, Class, etc. and reflectional realizations. Complex topics can be mono-form (mono-topics), consisting of a homogeneous distribution of one topic, or they can be poly-form, realizing a mixture of different topics. We will introduce first only mono-form topics, that is, distributions with only Booleans, or only Numerals, or only Lists, etc. Later poly-form topics (poly-topics) will be introduced which are showing interesting properties and relations between different topics. That is, in one complexion of ConTeXtures we will have Boolean, Numeric, List, etc. topics all being realized at once and interacting simultaneously. Therefore, a complex object $\text{c-ob}^{(m)}$, can be thematized at once as a Boolean and as a Numeric, etc. object.

Because architectonics, interactivity and reflectionality are prior to topics (data types, sorts, domains), topics are not on the top of the hierarchy of classification and presentation of ConTeXtures. On top are the design of architectonics, which here is constant and stable, considered as a linear mediation of systems, and then interactivity, which is studied in detail for some interactional modi. Next is reflectionality, which seems to be a new feature of polycontexturality, not yet well introduced before. The combination of both, interactivity and reflectionality as complementary features is also introduced. Locally, at each system, intra-contexturally, the well known feature of computation (Zuse, Turing, Church) and its iterative successivity is placed.

18.1 General pattern for (id, id, id)-modus of interaction

18.1.1 Proemiality of (mono-form) Boolean objects



samba((true, false), 3) =
define truei ord falsei, i=1,2,3
define true1 coinc true3,
define true3 coinc true1
define false1 exch true2,
define true2 exch false1
define false2 coinc false3,
define false3 coinc false2

More explicit:

samba (proem (true, false), 3))

$samba_i$ ($define_i$ $true_i$ $lambda_i$ (a b)
a)))
$samba_i$ ($define_i$ $false_i$ $lambda_i$ (a b)
b)))

samba (ARS, 3, true1true2true3)

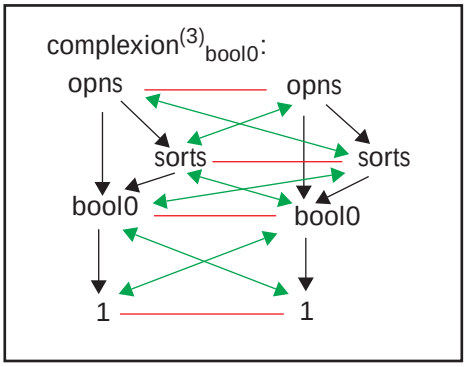
(define1 true1 (lambda (a b)
a)
(define1 false1 (lambda (a b)
b)
(define2 true2 (lambda (a b)
a)
(define3 true3 (lambda (a b)
a)

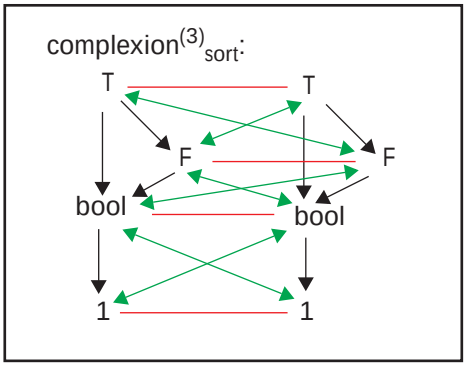
samba (ARS, 3, false1false2 false3)

(define1 false1 (lambda (a b)
b)
(define1 true2 (lambda (a b)
a)
(define2 false2 (lambda (a b)
b)
(define3 false3 (lambda (a b)
b)

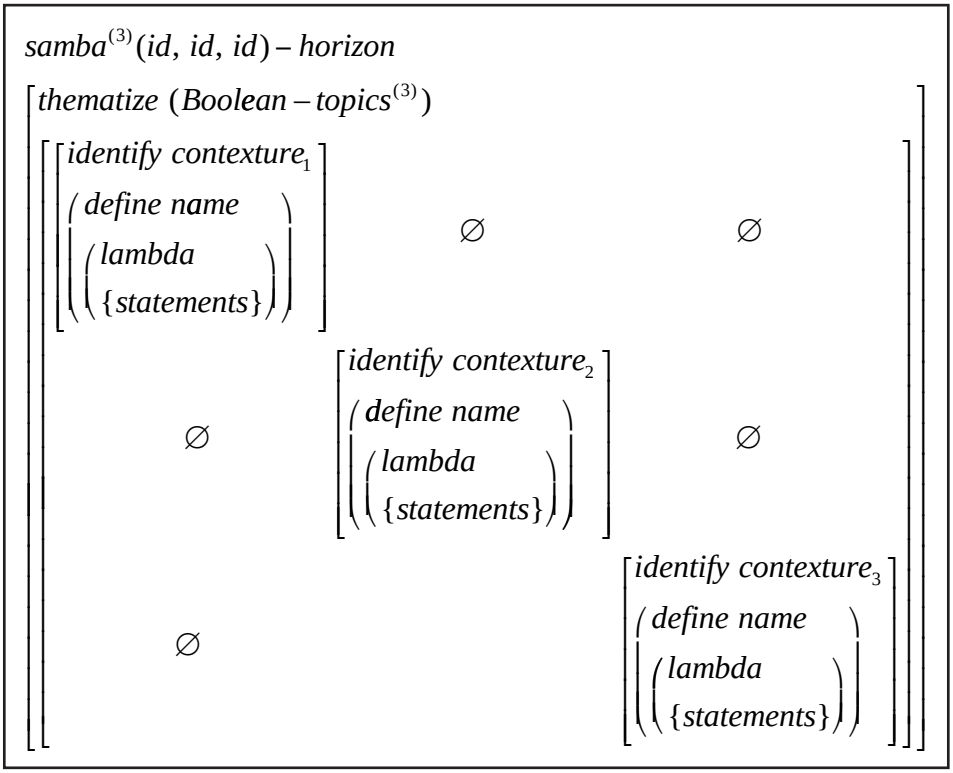
18.1.2 Proemiality of Booleans in abstract objects

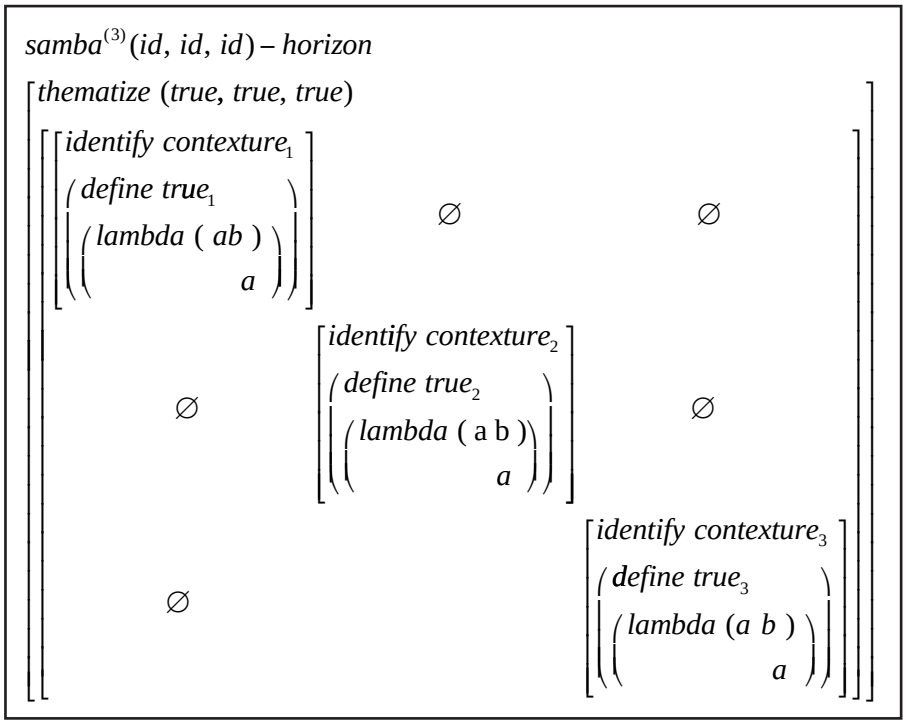
```
bool0=
    sorts
        bool
    opns
        T, F : —> bool
```

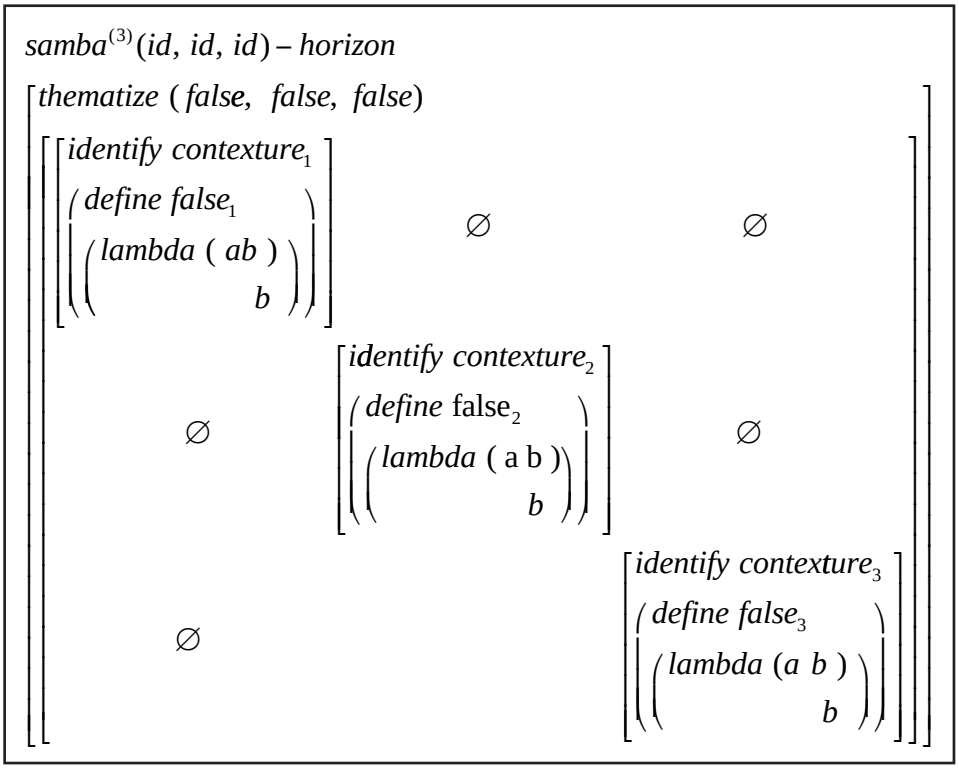




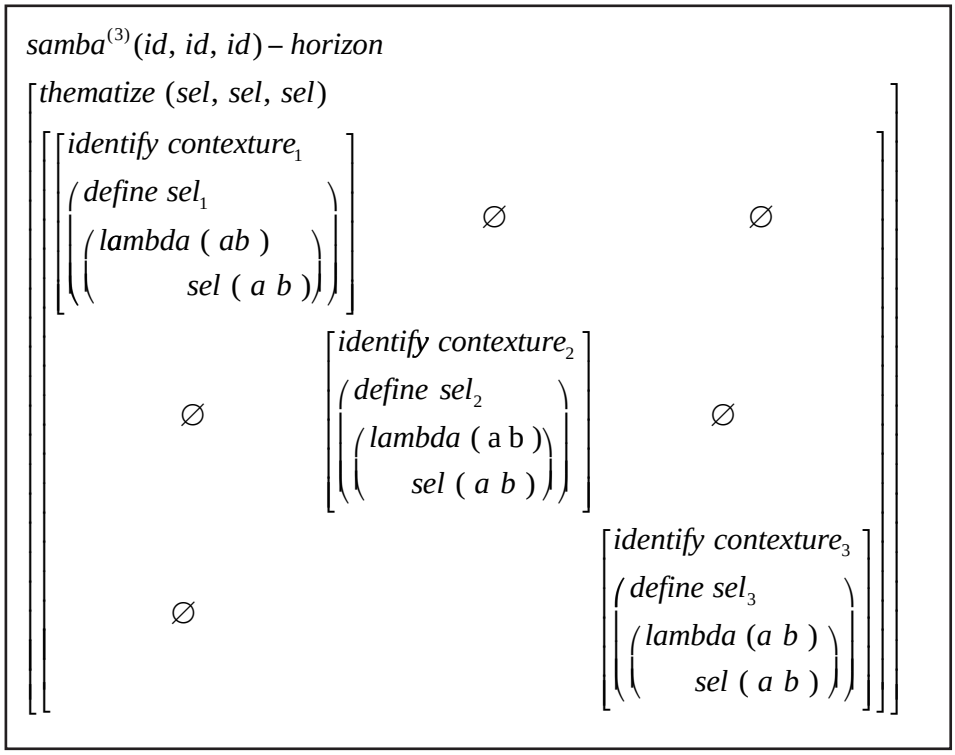
18.1.3 Samba of Boolean topics



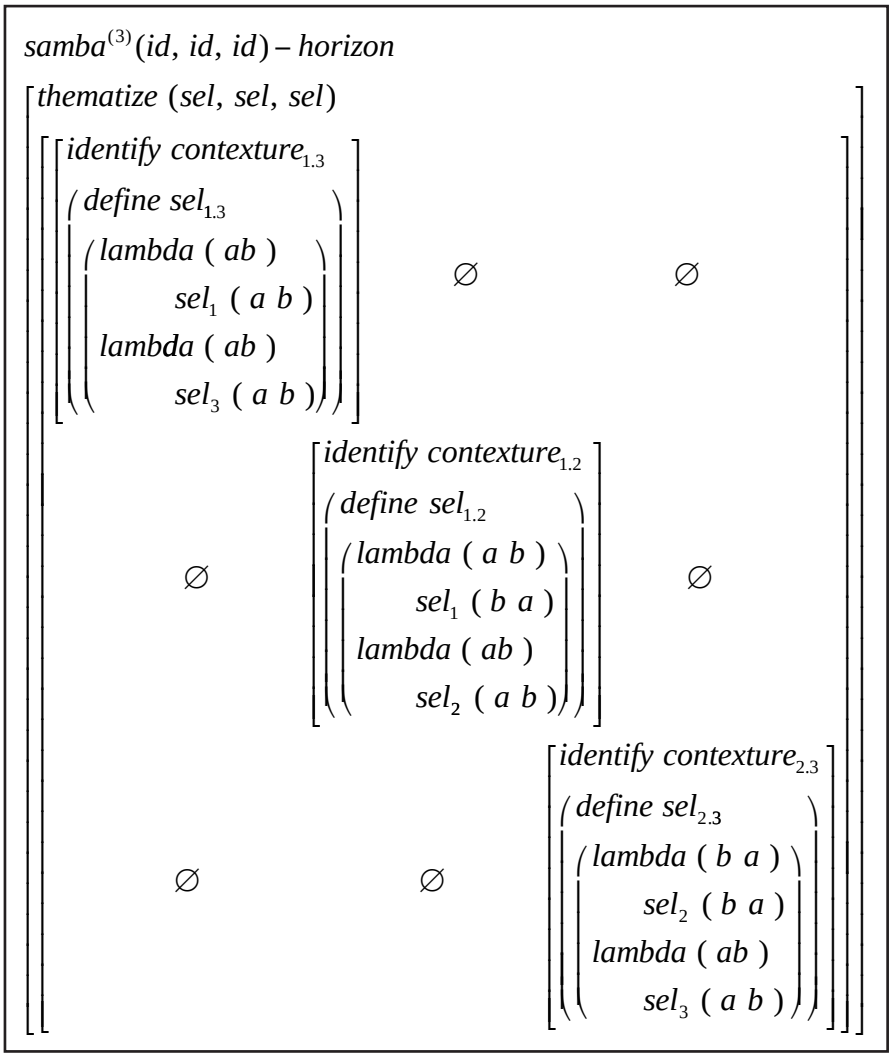

samba(3)(id, id, id) – horizon
thematize (true, true, true)
identify contexture1
define true1
lambda (ab)
a
∅
∅
∅
identify contexture2
define true2
lambda (a b)
a
∅
∅
identify contexture3
define true3
lambda (a b)
a

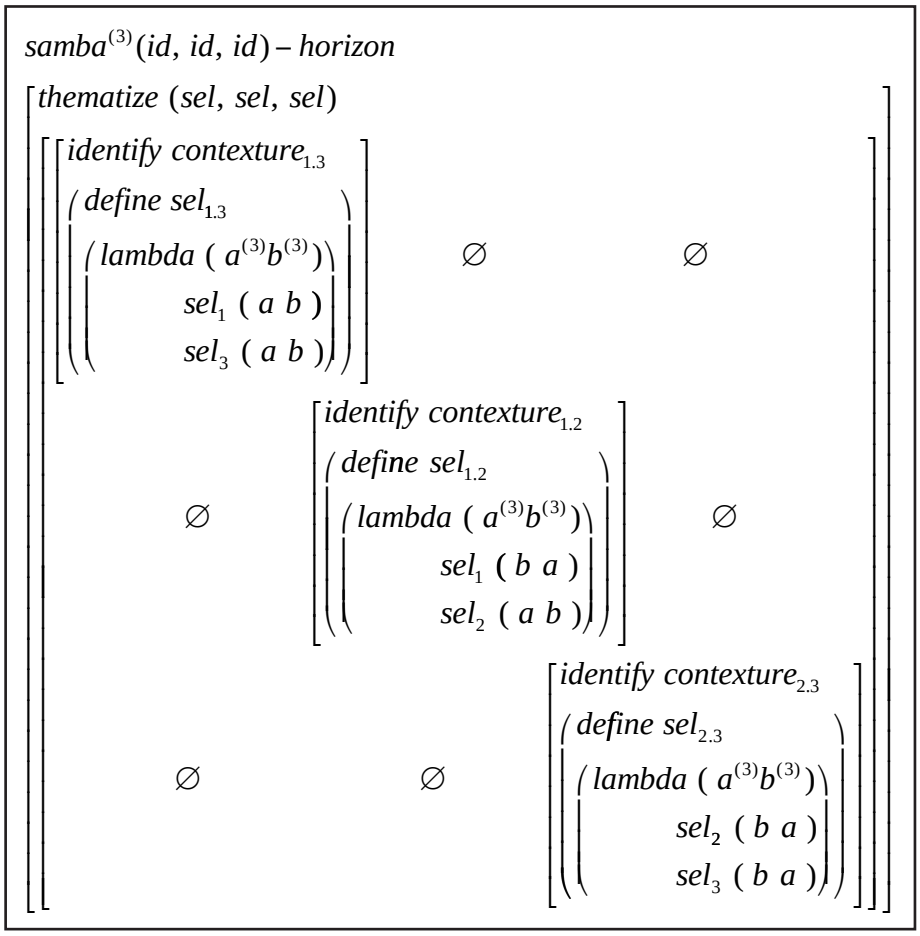

samba(3)(id, id, id) – horizon
thematize (false, false, false)
identify contexture1
define false1
lambda (ab)
b
∅
∅
∅
identify contexture2
define false2
lambda (a b)
b
∅
∅
identify contexture3
define false3
lambda (a b)
b

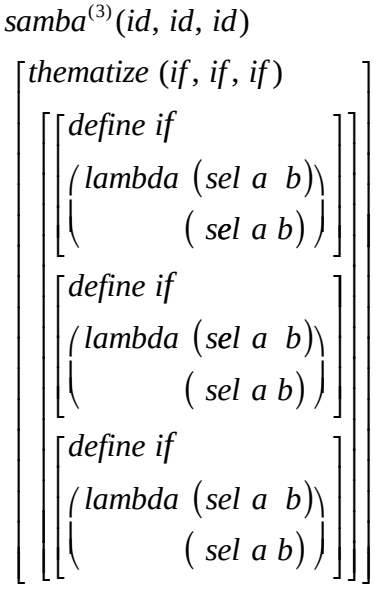
18.2 Poly-selectors in SAMBA'S



sel (a b) has to be replaced by (sel a b) !!


samba(3)(id, id, id) – horizon
thematize (sel, sel, sel)
identify contexture1.3
define sel1.3
lambda (ab)
sel1 (a b)
lambda (ab)
sel3 (a b)
∅
∅
∅
identify contexture1.2
define sel1.2
lambda (a b)
sel1 (b a)
lambda (ab)
sel2 (a b)
∅
∅
∅
identify contexture2.3
define sel2.3
lambda (b a)
sel2 (b a)
lambda (ab)
sel3 (a b)


samba(3)(id, id, id) – horizon
thematize (sel, sel, sel)
identify contexture1.3
define sel1.3
lambda (a(3)b(3))
sel1 (a b)
sel3 (a b)
∅
∅
∅
identify contexture1.2
define sel1.2
lambda (a(3)b(3))
sel1 (b a)
sel2 (a b)
∅
∅
∅
identify contexture2.3
define sel2.3
lambda (a(3)b(3))
sel2 (b a)
sel3 (b a)


samba(3)(id, id, id)
thematize (if, if, if)
define if
lambda (sel a b)
(sel a b)
define if
lambda (sel a b)
(sel a b)
define if
lambda (sel a b)
(sel a b)

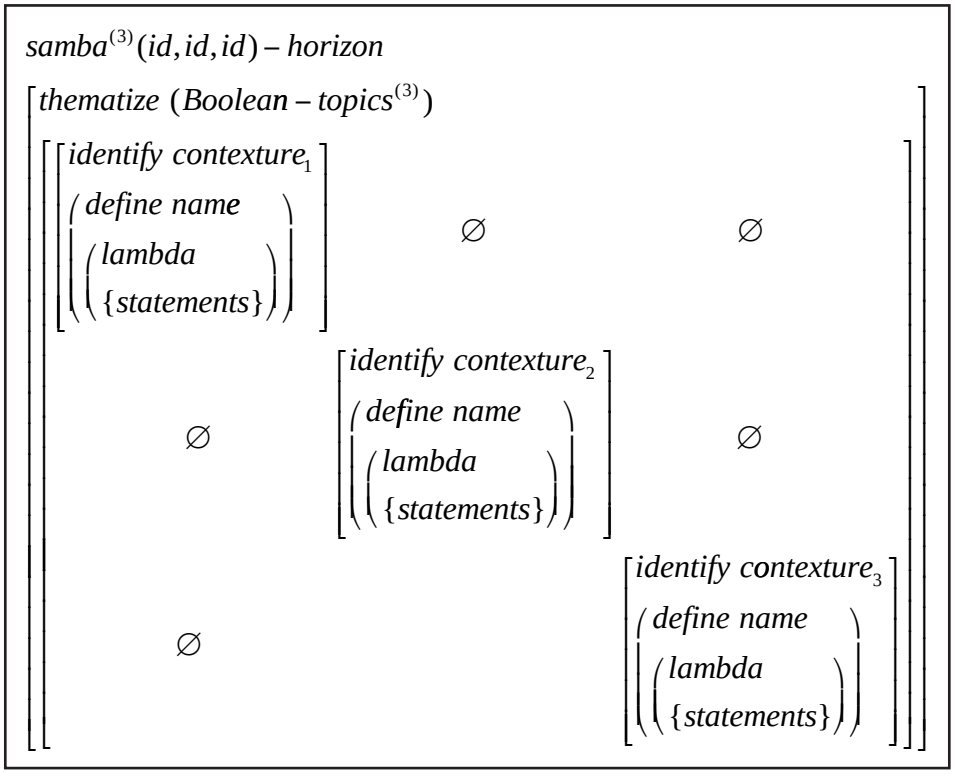
local conditional:
samba (ARS, 3; (if, if, ifi) :

```
( id1 (define if (lambda (sel a b)
            (sel a b))))

( id2 (define if (lambda (sel a b)
            (sel a b))))

( id3 (define if (lambda (sel a b)
            (sel a b))))
```

$$
\begin{array}{l}
samba^{(3)}(id, id, id) - horizon \\
\left[\begin{array}{l}
thematize\ (if, if, if) \\
\left[\begin{array}{ccc}
\left[\begin{array}{l} identify\ contexture_1 \\ \left(\begin{array}{l} define\ if_1 \\ \left(\begin{array}{l} lambda\ (\ ab\ ) \\ \qquad sel\ (\ a\ b\ ) \end{array}\right) \end{array}\right) \end{array}\right] & \varnothing & \varnothing \\
\varnothing & \left[\begin{array}{l} identify\ contexture_2 \\ \left(\begin{array}{l} define\ if_2 \\ \left(\begin{array}{l} lambda\ (\ \mathrm{a\ b}\ ) \\ \qquad sel\ (\ a\ b\ ) \end{array}\right) \end{array}\right) \end{array}\right] & \varnothing \\
\varnothing & & \left[\begin{array}{l} identify\ contexture_3 \\ \left(\begin{array}{l} define\ if_3 \\ \left(\begin{array}{l} lambda\ (a\ b\ ) \\ \qquad sel\ (\ a\ b\ ) \end{array}\right) \end{array}\right) \end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}
$$

18.3 Some application of basic Boolean abstractions

```
local:
(bdisp! truei)                              --> truei
(bdisp! flasei)
                          --> falsei
bdispl! = boolean display, takes the index of its arguments.

global:
(bdisp! true(3))
                          --> true1 coinc true3
                          --> true2 coinc false1
(bdisp! false(3))
                          --> false1 coinc true2
                          --> false2 coinc false3

isolated contitionals (yes/no, not logical conditionals)
( ifi truei
              (bdisp! truei)
              (bdisp! falsei)
                                       --> truei
( ifi falsei
              (bdisp! truei)
              (bdisp! falsei)
                                      --> falsei,     for all i=1,2,3

mediated conditionals of different strength:
( if111 true(3)
              (bdisp! true(3))
              (bdisp! false(3))
                                       --> true111
( if111 false(3)
              (bdisp! true(3))
              (bdisp! false(3))
                                       --> false111

( if113 true(3)
              (bdisp! true(3))
              (bdisp! false(3))
                                       --> true113
( if113 false(3)
              (bdisp! true(3))
              (bdisp! false(3))
                                       --> false113

( if133 true(3)
              (bdisp! true(3))
              (bdisp! false(3))
                                       --> true133
( if133 false(3)
              (bdisp! true(3))
              (bdisp! false(3))
                                       --> false133
```


samba(3)(id, id, id) – horizon
thematize (Boolean – topics(3))
identify contexture1
define name
lambda
{statements}
identify contexture2
define name
lambda
{statements}
identify contexture3
define name
lambda
{statements}

Formula for disjunction

samba (id, ARS, 3, (or, or, or))

Explicte introduction

samba (thematize (or, or, or) lambda$^{(3)}$ (a$^{(3)}$ b$^{(3)}$)

identify (define or1 (lambda (a b)
(if a a b))))
identify (define or2 (lambda (a b)
(if a a b)))
identify (define or3 (lambda (a b)
(if a a b)))

Short notation

samba (thematize **(or, or, or**) (lambda$^{(3)}$ (a$^{(3)}$ b$^{(3)}$)

(define$^{(3)}$ **or or or** (lambda$^{(3)}$ (a$^{(3)}$ b$^{(3)}$)

(if$^{(3)}$ a$^{(3}$ a$^{(3)}$ b$^{(3)}$)))))

samba (id, ARS, 2, (or, and, or))

((define or1 (lambda (a b)
(if a a b))))
((define or2 (lambda (a b)
(if a b a))))
((define or3 (lambda (a b)
(if a a b))))

18.3.1 General pattern for (id, id, id)-modus of binary junctional interaction

$$samba^{(3)}(id,id,id)$$

$$\begin{bmatrix} thematize\ (topics^{(3)}) \\ \begin{bmatrix} \begin{bmatrix} identify\ contexture_1 \\ \begin{bmatrix} define\ junction \\ \begin{bmatrix} lambda\ (\ a\ b\ ) \\ \ \ if\ \ (cond-\ a\ b\ ) \end{bmatrix} \end{bmatrix} \end{bmatrix} & \varnothing & \varnothing \\ \varnothing & \begin{bmatrix} identify\ contexture_2 \\ \begin{bmatrix} define\ junction \\ \begin{bmatrix} lambda\ (\ a\ b\ ) \\ \ \ if\ \ (cond-\ a\ b\ ) \end{bmatrix} \end{bmatrix} \end{bmatrix} & \varnothing \\ \varnothing & & \begin{bmatrix} identify\ contexture3 \\ \begin{bmatrix} define\ junction \\ \begin{bmatrix} lambda\ (\ a\ b\ ) \\ \ \ if\ \ (cond-\ a\ b\ ) \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Sup-operators: (id, id, id)
Set of junctions: { and, or }

Condition for conjunction and disjunction, cond-(a b)

condition for conjunction: (if a b a)
condition for disjunction: (if a a b)

Combinations of junctions

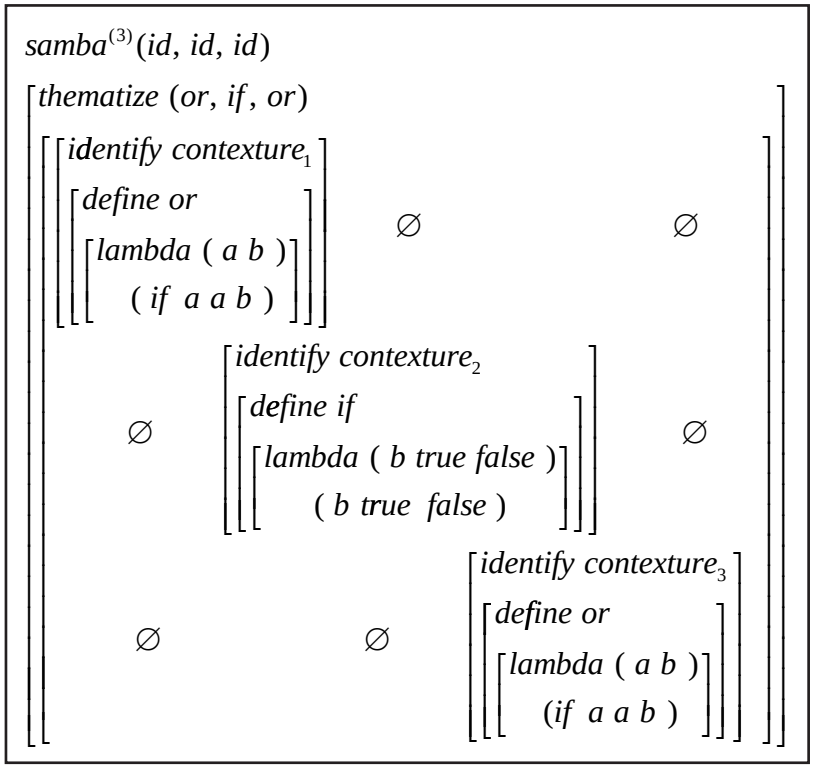
{(or, or, or)
(or, or, and)
(or, and, or)
(or, and, and)
(and, or, or)
(and, or, and)
(and, and, or)
(and, and, and)}

$$samba^{(3)}(id, id, id)$$

$$\begin{bmatrix} thematize\ (or,\ and,\ or) & & \\ \begin{bmatrix} \begin{bmatrix} identify\ contexture_1 \\ \begin{bmatrix} define\ or \\ \begin{bmatrix} lambda\ (\ a\ b\ ) \\ if\ \ (a\ a\ b\ ) \end{bmatrix} \end{bmatrix} \end{bmatrix} & \varnothing & \varnothing \\ \varnothing & \begin{bmatrix} identify\ contexture_2 \\ \begin{bmatrix} define\ and \\ \begin{bmatrix} lambda\ (\ a\ b\ ) \\ if\ \ (a\ b\ a\ ) \end{bmatrix} \end{bmatrix} \end{bmatrix} & \varnothing \\ \varnothing & & \begin{bmatrix} identify\ contexture3 \\ \begin{bmatrix} define\ or \\ \begin{bmatrix} lambda\ (\ a\ b\ ) \\ if\ \ (a\ a\ b\ ) \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

18.3.1.1 Non-realizable combinations

Why is the combination (or1, if2, or3) not realizable in the chosen setting?



The question of realizability/non-realizabiltty of function, shows that on a meta-language level we have not only to distinguish the different logical values and their loci of distribution but also the difference compatibility/non-compatibility defined by the rules of mediation.

18.4 Violating the mediation rules

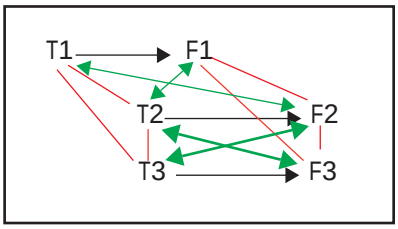
The easiest way to demonstrate the incompatibility of the combination we can ask for its Boolean representation. This approach can be generalized and applied to c-obs in general delivering a method to check compatibility and realizability of mediations, esp. of poly-topic mediations.

```
( id, id, id )
( bdisp! ( or1, if2, or3 )
( bdisp! or )
( bdisp! if )
( bdisp! or )

( bdisp! or)
( bdsp! or true1 true1)  --> true1
( bdsp! or true1 false1)  --> true1
( bdsp! or false1 true1)  --> true1
( bdsp! or false1 false1)  --> false1

( bdisp! if )
( bdsp! if true2 true2)  --> true2
( bdsp! if true2 false2)  --> false2
( bdsp! if false2 true2)  --> true2
( bdsp! if false2 false2)  --> true2

( bdisp! or)
( bdsp! or true3 true3)  --> true3
( bdsp! or true3 false3)  --> true3
( bdsp! or false3 true3)  --> true3
( bdsp! or false3 false3)  --> false3
```

If we confront this result with the proemial conditions of our truth values, esp. false2 coinc false3, we observe a collision between the two results:

```
( bdsp! if false2 false2)  --> true2
( bdsp! or false3 false3)  --> false3
```

The values true2/false3 are neither equal nor analog, they are different and belong to the exchange relation and not the coincidence relation, like, say true1/true3 or false1/true2. Therefore, the mediation of the combination (or1, if2, or3) is not realized.

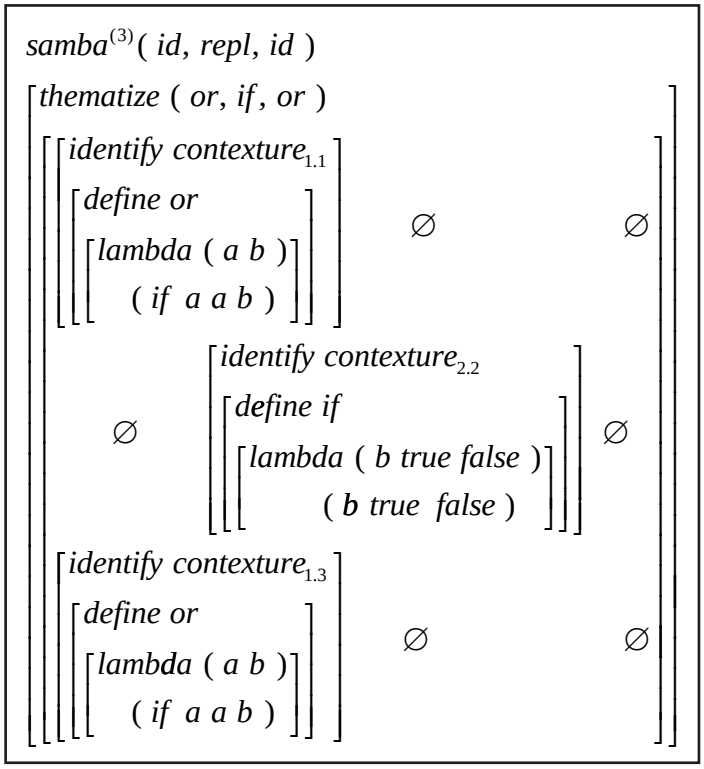


samba((true, false), 3) =
define truei ord falsei, i=1,2,3
define true1 coinc true3
define false1 exch true2
define false2 coinc false3

This type of consideration has to be applied also to all other topics and their conditions of mediation outside the range of Boolean values.

Another mediation

The combination (or, if, or) can be realized under the pattern [id, id, repl], that is, in a different category of the interaction/reflexion pattern.



Template: [(S103), (S020), (S000)]

```
( id, id, repl )( or1, if2, or1 )

( bdisp! ( or1, if2, or1 )
( bdisp! or)
( bdsp! or true1 true1)    --> true1
( bdsp! or false1 false1)  --> false1

( bdisp! if )
( bdsp! if true2 true2)    --> true2
( bdsp! if false2 false2)  --> true2

( bdisp! or)
( bdsp! or true3 true3)  --> true1
( bdsp! or repl (false3 false3))  --> false1
                                  (replication of or3 as or1)
```

Now, we observe no collision between the two results, true2/false1, therefore this type of combination is realizable.

18.4.1 Typology of Boolean binary combinations

c.f. Morphogrammatik

18.5 Application of extended logical abstractions

```
local negations
(bdisp! (truei))
                    --> truei

(bdisp! (falsei))
                    --> false1
(bdisp! (true(3)))
                    --> (true1 true2 true3)

(bdisp! (fals(3)))
                    --> ( false1 false2 fals3)

(bdisp! (noti truei))
                            --> falsei

(bdisp! (noti falsei))
                            --> truei, i= 1,2

mediated negations
(bdisp! (not1 true(3)))
                            --> (false1, true3, true2)

(bdisp! (not1 false(3)))
                            --> (true1, false3, false2)

(bdisp! (not2 true(3)))
                            --> ( true3, false2, true1)

(bdisp! (not2 false(3)))
                            --> ( false3, true2, false1)
```

Junctions, mono- and polyform

```
(bdisp! (and(3) true(3) true(3)))
                            --> (true1, true2, true3)

(bdisp! (and(3) false(3) false(3)))
                            --> (false1, false2, false3)

(bdisp! (and(3) true(3) false(3)))
                            --> (false1, false2, false3)

(bdisp! (and(3) false(3) true(3)))
                            --> (false1, false2, false3)

(bdisp! (or(3) true(3) true(3)))
                            --> (true1, true2, true3)

(bdisp! (or(3) false(3) false(3)))
                            --> (false1, false2, false3)

(bdisp! (or(3) true(3) false(3)))
                            --> (true1, true2, true3)

(bdisp! (or(3) false(3) true(3)))
```

```
                          --> (true1, true2, true3)

(bdisp! ((and,or,or) true(3) true(3)))
                          --> (true1, true2, true3)

(bdisp! ((and,or,or) false(3) false(3)))
                          --> (false1, false2, false3)

(bdisp! ((and,or,or) true(3) false(3)))
                          --> (false1, true2, true3)

(bdisp! ((and,or,or) false(3) true(3)))
                          --> (false, true2, true3)
```

Junctions plus transjunctions

```
(bdisp! ((trans,or,and) true(3) true(3)))
                          --> (true1, true2, true3)

(bdisp! ((trans,or,and) false(3) false(3)))
                          --> (false1, false2, false3)

(bdisp! ((trans,or,and) true(3) false(3)))
                         --> ( ( empt1 , false2, false3), true2, false3)

(bdisp! ((trans,or,and) false(3) true(3)))
                          --> (( empt1 , false2, false3), true2, false3)
```

18.6 Distribution of Distributive Boolean Lattices

Under the strict id-modus of distribution a strict *logical parallelism* without negations can be introduced. Lattices are important for knowledge representation (conceptual graphs, Sowa, Wille) and data bases. Obviously, their distribution involves a radical paradigm change in dealing with knowledge representations and data bases. An object can appear as conjunctive and simultaneously as disjunctive depending of the viewpoint of thematization, reflecting its ambiguity in a third mediating system.

Distributing distributive latices

Identity: X * X == X

Commutativity: X * Y == Y * X, with *, + = {and, or}

Associativity: X * (Y * Z) == (X * Y) * Z

Absorbtion: X * (X + Y) == X,

Distributivity: X * (Y + Z) == (X * Y) + (X * Z)

Modus ponens: X, X impl Y ==> Y

To introduce a Boolean lattice we would have to leave the strict id-pattern and involve additionally the super-operator perm.

$L_{distr} = (com, ass, abs, distr)$

$L^{(3)}_{distr} : (L^{1}_{distr}, L^{2}_{distr}, L^{3}_{distr}) \longrightarrow L^{(3)}_{distr}$

Some balanced examples

3-Identity: $X^{(3} *** X^{(3} == X^{(3)}$ with *, + = {and, or}

3-Commutativity:
X *** Y == Y *** X
X*+* Y == Y *+* X

3-Associativity:
X *** (Y *** Z) == (X *** Y) *** Z
X *++ (Y *++ Z) == (X *++ Y) *++ Z
X *+* (Y *+* Z) == (X *+* Y) *+* Z

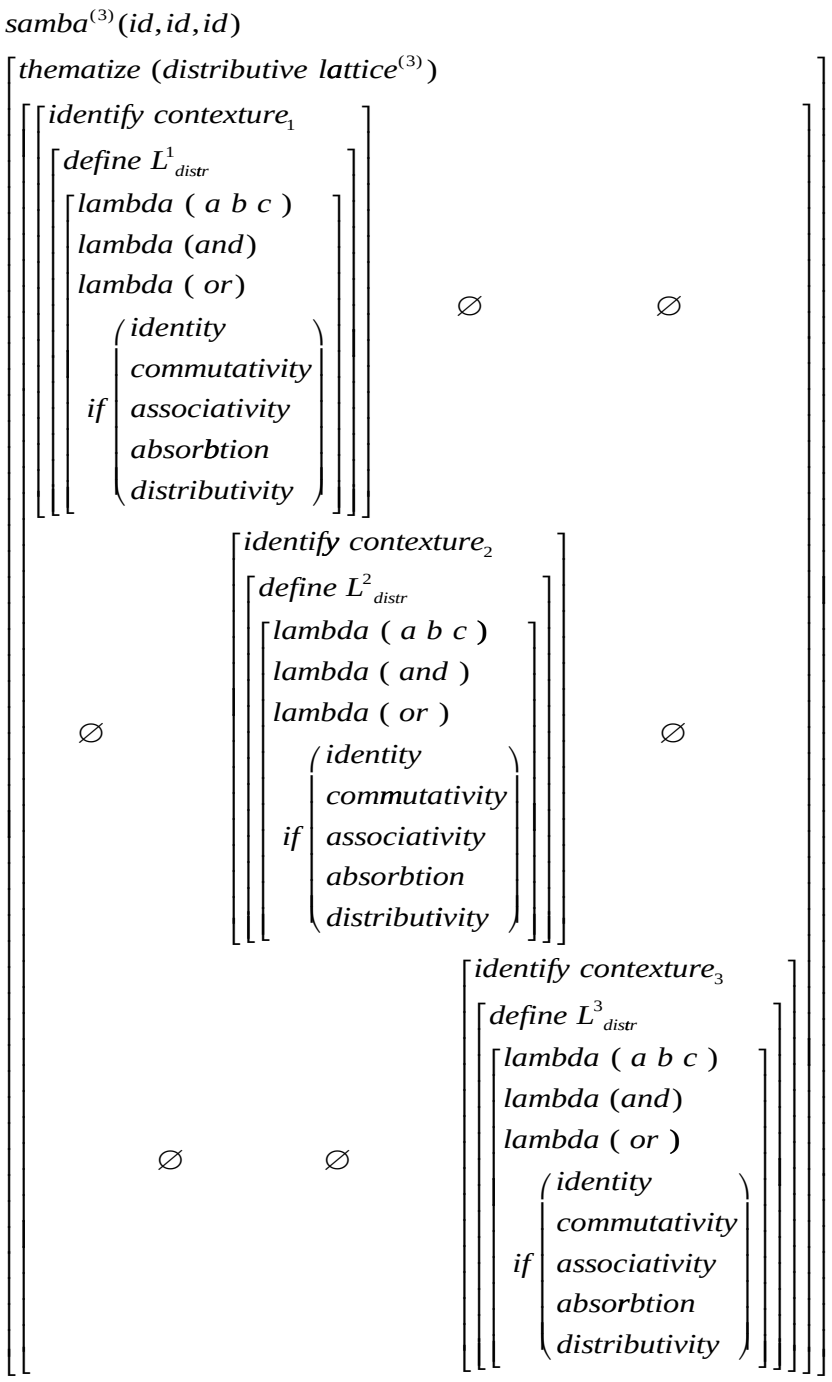
3-Absorbtion:
X *** (X +++ Y) == X
X *+* (X +*+ Y) == X

3-Distributivity:
X *** (Y +++ Z) == (X *** Y) +++ (X *** Z)
X *++ (Y + **Z) == (X *++ Y) +** (X *++ Z)
X +*+ (Y *+* Z) == (X +*+ Y) *+* (X +*+ Z)

3-Modus ponens: $X^{(3)}, X^{(3)}\ impl^{(3)}\ Y^{(3)} ==>^{(3)} Y^{(3)}$

Models

As in the mono-contextural case of lattices interesting models in a corresponding polycontextural set theory can be studied and applied.



Template: [(S100), (S020), (S003)]
Pattern: (id, id, id)
Topic: Boolean

Poly-Arithmetical Topics

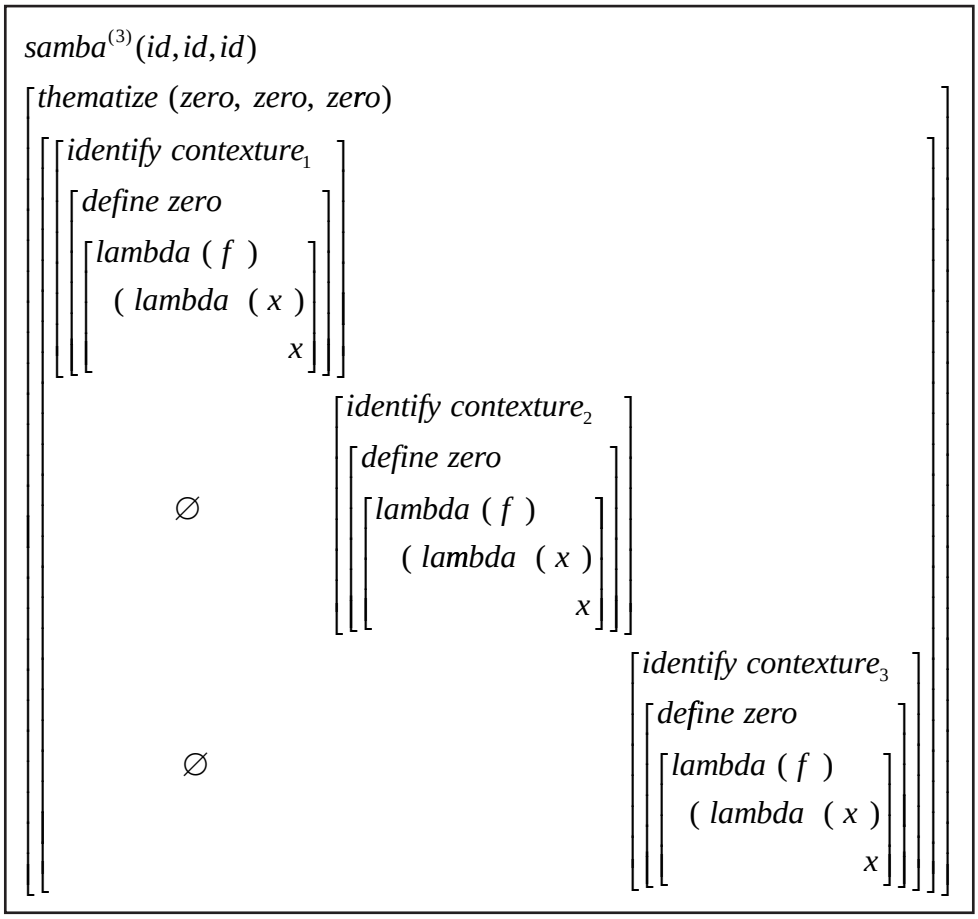
$$samba^{(3)}(id,id,id) - horizon$$

$$\left[thematize\ (arithmetical - topics^{(3)}) \right.$$

$$\left[\left[\begin{bmatrix} identify\ contexture_1 \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} \quad \varnothing \quad \varnothing \right. \right.$$

$$\varnothing \quad \begin{bmatrix} identify\ contexture_2 \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} \quad \varnothing$$

$$\left. \left. \varnothing \quad \begin{bmatrix} identify\ contexture_3 \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} \right] \right]$$



samba (ARS, 3, zero)

```
( define zero (lambda ( f )
                ( lambda  ( x )
                  x ))
( define zero (lambda ( f )
                ( lambda  ( x )
                  x ))
( define zero (lambda ( f )
                ( lambda  ( x )
                  x ))
```

$$samba^{(3)}(id, id, id)$$

$$\left[\begin{array}{l} thematize\ (zero,\ zero,\ zero) \\ \left[\begin{array}{l} \left[\begin{array}{l} identify\ contexture_1 \\ \left[\begin{array}{l} define\ zero \\ \left[\begin{array}{l} lambda\ (\ f\ ) \\ \left[\begin{array}{l} (\ lambda\ \ (\ x\ ) \\ \qquad\qquad x \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \\ \left[\begin{array}{l} identify\ contexture_2 \\ \left[\begin{array}{l} define\ zero \\ \left[\begin{array}{l} lambda\ (\ f\ ) \\ \left[\begin{array}{l} (\ lambda\ \ (\ x\ ) \\ \qquad\qquad x \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \\ \left[\begin{array}{l} identify\ contexture_3 \\ \left[\begin{array}{l} define\ zero \\ \left[\begin{array}{l} lambda\ (\ f\ ) \\ \left[\begin{array}{l} (\ lambda\ \ (\ x\ ) \\ \qquad\qquad x \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

samba (id, ARS, 3, one)

```
( define one  ( lambda ( f )
                    ( lambda ( x )
                        ( f x )))
( define one  ( lambda ( f )
                    ( lambda ( x )
                        ( f x )))
( define one  ( lambda ( f )
                    ( lambda ( x )
                        ( f x )))
```

sambba (id, ARS, 3, three)

```
( define one  ( lambda ( f )
                    ( lambda ( x )
                      f (f ( f x ))))))
( define one  ( lambda ( f )
                    ( lambda ( x )
                      f (f ( f x ))))))
( define one  ( lambda ( f )
                    ( lambda ( x )
                      f (f ( f x ))))))
```

samba (id, ARS, 3, zerop)

```
( define zerop1 ( lambda ( n )
                    (( n ( lambda ( y )
                             false1 ))
                     true1 )))
( define zerop2 ( lambda ( n )
                    (( n ( lambda ( y )
                             false2 ))
                     true2 )))
( define zerop3 ( lambda ( n )
                    (( n ( lambda ( y )
                             false3 ))
                     true3 )))
```

samba (id1, ARS, 3, add)

```
        ( define add1   ( lambda (m n )
                          ( lambda ( f )
                              ( compose (m f ) (n f ))))
         ( define add2   ( lambda (m n )
                          ( lambda ( f )
                           ( lambda ( f )
                              ( compose (m f ) (n f ))))
         ( define add3   ( lambda (m n )
                          ( lambda ( f )
                           ( lambda ( f )
                              ( compose (m f ) (n f ))))
```

samba (id, ARS, 3, succ)

```
( define  succ1    ( lambda (n )
                       ( lambda ( f )
                          ( compose1  f ( n f ))))))
 ( define  succ2   ( lambda (n )
                       ( lambda ( f )
                          ( compose2  f ( n f ))))))
 ( define  succ3    ( lambda (n )
                       ( lambda ( f )
                          ( compose3  f ( n f ))))))
```

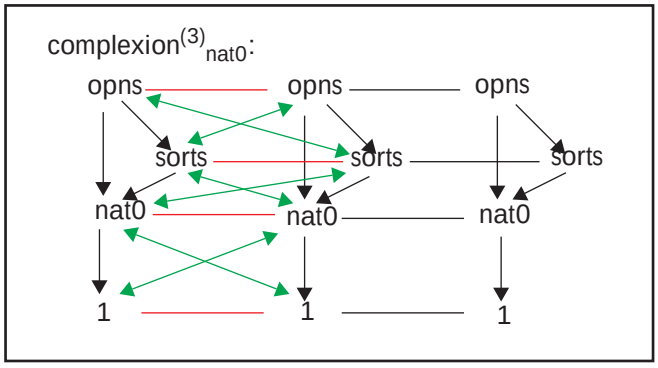
18.6.1 Second order numerical functions

samba (id, ARS, 3, compose)

```
( define compose1   ( lambda ( f g )
                        ( lambda ( x )
                            (f (g x ))))
( define compose2   ( lambda ( f g )
                        ( lambda ( x )
                            (f (g x ))))
( define compose3   ( lambda ( f g )
                        ( lambda ( x )
                            (f (g x ))))
```

18.6.2 Dissemination of numeric objects

As an application of the idea of proemiality I introduce the dissemination of 3 abstract objects nat0.



(zero —> (nat —> nat)) —> nat0

dissemination$^{(3)}_{nat0}$

DISS$^{(3)}$ (object $_{nat0}$) = object1$_{nat0}$ § object2$_{nat0}$ § object3$_{nat0}$

object1$_{nat0}$ —> object1$_{nat0}$
object2$_{nat0}$ —> object2$_{nat0}$
object3$_{nat0}$ —> object3$_{nat0}$

Natural system:
nat0$^{(3)}$ = nat0(1) § nat0(2) § nat0(3)

Sorts:
sorts$^{(3)}$ = sorts(1) § sorts(2) § sorts(3)

Operations:
opns$^{(3)}$ = opns(1) § opns(2) § opns(3)

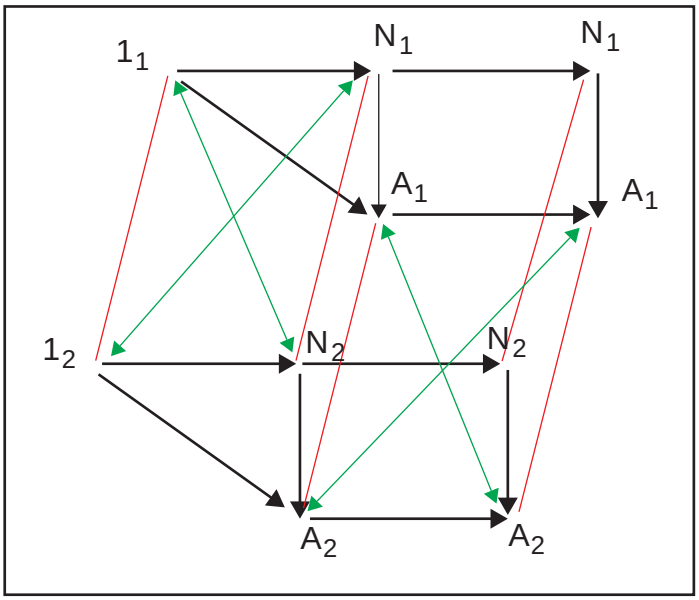
zero$^{(3)}$: —> nat$^{(3)}$ with
zero1: —> nat1
zero2: —> nat2
zero3: —> nat3 and

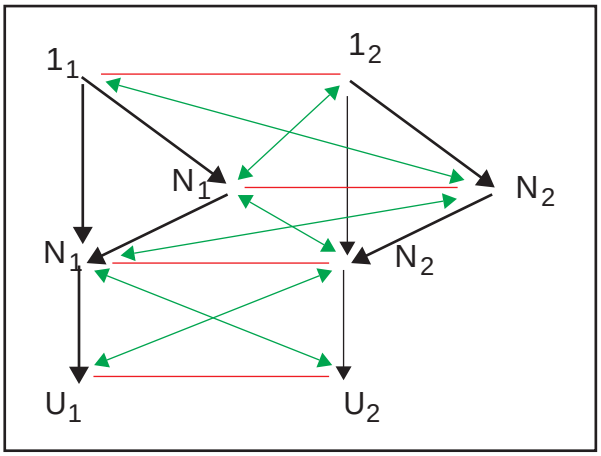
suc$^{(3)}$: nat$_{(3)}$ —> nat$^{(3)}$ with
nat1 —> nat1
nat2 —> nat2
nat3 —> nat3

This triple clone of the above natural system produces naturally infinite series of expressions of the following form if we apply the operations in a parallel way.

zero, suc(zero), suc(suc(zero)), suc(suc(suc(zero))), ...
zero, suc(zero), suc(suc(zero)), suc(suc(suc(zero))), ...
zero, suc(zero), suc(suc(zero)), suc(suc(suc(zero))), ...

In another notation of this results in 3-tuples of terms.
(zero, zero, zero), (suc(zero), suc(zero), suc(zero)), ...





18.6.3 Super-operators in disseminated numeric systems

$DISS^{(3)}$ (object $_{nat0}$) = $object1_{nat0}$ § $object2_{nat0}$ § $object3_{nat0}$

$object1_{nat0}$ $_{-->}$ $object1_{nat0}$
$object2_{nat0}$ $_{-->}$ $object2_{nat0}$
$object3_{nat0}$ $_{-->}$ $object3_{nat0}$

object1 $_{-->}$ object1
object2 $_{-->}$ object2
object3 $_{-->}$ object3

```
ID(3) (zero(3)) : --> nat(3)
zero1:  --> nat1
zero2:  --> nat2
zero3:  --> nat3

(ID PERM2 PERM3) zero(3) : --> nat(3) with
zero1:  --> nat1
zero2:  --> nat3
zero3:  --> nat2

(ID BIF1,3 ID)zero(3) : --> nat(3) with
zero1:  --> nat1 simul nat2
zero2:  --> nat2
zero3:  --> nat3 simul nat2

(RED2 ID ID) zero(3) : --> nat(3) with
zero1:  --> nat2
zero2:  --> nat2
zero3:  --> nat3
```

In contrast to the purely parallel construction we have to introduce a more complex notation for the general case.

Until now I have treated zero as an object and different types of zeros as objects belonging to different contextures. This is a quite conservative introduction. In correspondence to the idea of proemiality and polycontexturality, it is more appropriate to think of zero as an action. In this sense zero is the notation of the action of beginning. There are many beginnings but no single origin. Actions, and especially simultaneous actions, are not necessarily connected with the notion of identity. In contrast, objects are very close to the notion of identity. The classical concept of an object coincides more or less with this notion of identity. Actions are not given as ontological entities, therefore they have to be interpreted. An interpretation involves an interpreter, which is a point of view. Because there is no single privileged point of view there is a multitude of interpreters, interpreting successively or simultaneously the realizations of actions.

The Operator (ID BIF1,3 ID)zero(3) suggests that there are additionally to the genuine objects of the systems objects from the neighbor systems too.

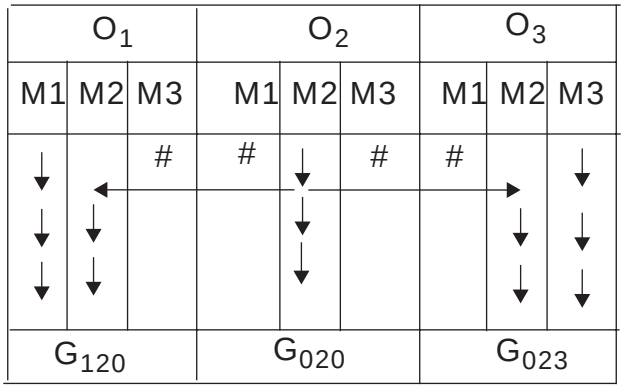
The first object of the application of the operator (ID BIF1,3 ID) to zero(3) is therefore:

[(zero, zero, #), (#, zero, #) (#, zero, zero)]

$$\frac{(ID,\ BIF^{1,3},\ ID)\ (OP1,\ OP2,\ OP3)}{(OP1,\ OP2,\ \#)\ (\#,\ OP2,\ \#)\ (\#,\ OP2,\ OP3)}$$

In this sense the Operator (ID, $BIF^{1,3}$, ID) is only a perhaps misleading abbreviation of the above more explicit notation.

Diagramm 7 **Transition-Diagramm of the Operation (ID BIF1,3 ID)**



List-Topics

Lists as a data type are defined as compositions of ordered pairs, and generally of n-tuples. A pair of a list is composed of a head and a tail. The head points to a data item and the tail points to the next pair in the list. The tail in the last pair of the list points to a special element called "nil". To be able to speak of a head and a tail of a pair is possible only because the order between head and tail is pre-given, it is not really constructed by a lambda abstraction. The situation is not better as in mathematics where the ordered pair is introduced axiomatically presupposing some tricks which are intuitively acceptable and are inventing a useful convention based on cultural agreements.

From A++

Lists are implemented as linked lists of pairs. A pair is composed of a head and a tail. The head points to a data item and the tail points to the next pair in the list.
In order to work with pairs and lists the following abstractions are needed as a minimum:

* **cons** the constructor of a **pair**
* **car** the selector of the **head** of a pair
* **cdr** the selector of the **tail** of a pair
* **nullp** a predicate to check, whether the list is **empty**
* **pairp** a predicate to check, whether the object is a pair.

```
( define cons  (lambda (hd tl)
                        ( lambda ( sel )
                             ( sel hd tl ))))
(define  nil (lambda (f)
                    true ))

( define car  (lambda ( l )
                     ( l true )))

( define cdr  (lambda (l)
                     ( l false )))
( define nullp  ( lambda (l)
                      ( l (lambda ( hd tl)
                                false ))))
( define length  (lambda (l)
                           (if (nullp l)
                                 zero
                                   (add one ( length ( cdr 1))))))
( define remove
    ( lambda ( obj l )
       ( if ( nullp l )
            nil
            ( if ( equalx obj ( car l )
             ( remove obj ( cdr l )
              (( cons ( car l )  ( remove obj ( cdr l )))))))
( define  nth
  ( lambda ( n l )
    ( if ( equaln n one )
       ( car l )
        ( nth ( sub n one ) ( cdr l )))))
```

http://www.aplusplus.net/bookonl/

Mono-form operators are of the form (op1=op2=…=ops). They are homogene in respect of their type of operation.

$$samba^{(3)}(id,id,id) - horizon$$

$$\begin{bmatrix} thematize\ (list - topics^{(3)}) & & \\ \begin{bmatrix} \begin{bmatrix} identify\ contexture_1 \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} & \varnothing & \varnothing \\ \varnothing & \begin{bmatrix} identify\ contexture_2 \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} & \varnothing \\ \varnothing & & \begin{bmatrix} identify\ contexture_3 \\ \begin{pmatrix} define\ name \\ \begin{pmatrix} lambda \\ \{statements\} \end{pmatrix} \end{pmatrix} \end{bmatrix} \end{bmatrix} & \end{bmatrix}$$

```
samba ( thematize (cons, cons, cons)   lambda (3) (hd(3)  tl(3) )
                                  (define cons1  (lambda (hd tl)
                                      (lambda ( sel )
                                          ( sel hd tl ))))
                                   (define cons2  (lambda (hd tl)
                                      (lambda ( sel )
                                          ( sel hd tl ))))
                                    (define cons3 ( lambda (hd tl)
                                      (lambda ( sel )
                                          ( sel hd tl )))))

samba ( thematize (nil, nil, nil)   lambda (3) (f(3))
                                (define nil ( lambda (f1)
                                    true1))
                                 (define nil  ( lambda (f2)
                                    true2))
                                 (define nil lambda ( lambda (f3)
                                    true3)))))
samba ( thematize (cdr, cdr, cdr)  ( lambda (3) ( l(3))
                                    ( define cdr (lambda ( l )
                                                        ( l false)
                                                ( define cdr (lambda ( l )
```

```
                                    ( l false)
                                ( define cdr (lambda ( l )
                                    ( l false)))))
```

```
samba ( thematize (car, car, car)  ( lambda (3) ( l(3))
                        ( define car (lambda ( l )
                                    ( l true)
                                ( define car (lambda ( l )
                                    ( l true)
                                ( define car (lambda ( l )
                                    ( l true)))))
```

Short explicite notations:

samba((3), (id, id, id))

(thematize **(car, car, car**) (lambda$^{(3)}$ (l$^{(3)}$))

(define$^{(3)}$ **car car car** (lambda$^{(3)}$ (l$^{(3)}$)

(l$^{(3)}$ true$^{(3)}$)))))

$$samba^{(3)}(id,\ id,\ id)$$
$$\left[\begin{array}{l} thematize\ (car,\ car,\ car) \\ \left[\begin{array}{l} lambda^{(3)}\ (\ l^{(3)}) \\ \left[\begin{array}{l} define^{(3)}\ (car,\ car,\ car) \\ \left(\left(lambda^{(3)}\ (\ l^{(3)})\right)\right. \\ \left.\left(\ (\ l^{(3)}\ true^{(3)})\right)\right. \end{array}\right] \end{array}\right] \end{array}\right]$$

Poly-form operators are of the form (op1/=op2=.../=ops). They are homogene in respect of their type of operation.

```
samba ( thematize (nil, cons, car)   lambda (3) (f(3))
                        (define nil1 ( lambda (f1)
                                    true1))
                        (define cons2  (lambda (hd tl)
                            (lambda ( sel )
                                ( sel hd tl ))))
                        (define car3  ( lambda ( l )
                                    ( l true3 )))
```

$$samba^{(3)}(\ id,\ id,\ id\ )$$

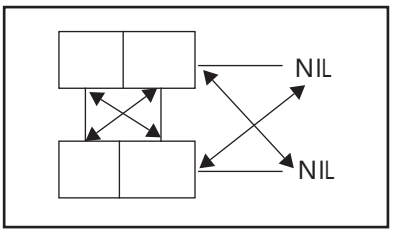
$$\left[\begin{array}{l} thematize\ (nil,\ cons,\ car) \\ \left[\begin{array}{l} \left[\begin{array}{l} define\ nil \\ \left(\begin{array}{ll} lambda & (\ f\ ) \\ & true \end{array}\right) \end{array}\right] \\ \left[\begin{array}{l} define\ cons \\ \left(\begin{array}{ll} lambda & (\ hd\ tl\ ) \\ & \left(\begin{array}{ll} lambda & (\ sel\ ) \\ & (sel\ hd\ tl\ ) \end{array}\right) \end{array}\right) \end{array}\right] \\ \left[\begin{array}{l} define\ car \\ \left(\begin{array}{ll} lambda & (\ l\ ) \\ & (l\ true) \end{array}\right) \end{array}\right] \end{array}\right] \end{array}\right]$$

18.6.4 Proemiality of poly-list operators

The basic concepts in SAMBA'S are not binary, but on the base of their proemiality, there are ternary, quadruples or genuine, not reducible compositions of pairs. The proemiality in poly-lists is strict structural and is not concerned with specific objects or elements of the lists.

samba (pair, 1) = (pair)
samba (list, 1) = (list)

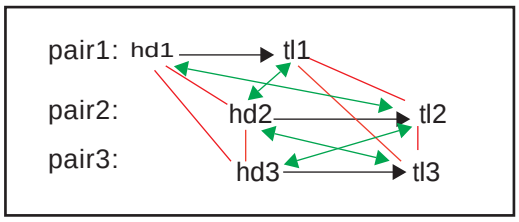
samba (pair, 3)
samba (pair)
samba (pair)
samba (pair)



car (pair1) coinc car (pair2)
car (pair1) coinc car (pair3)
car (pair1) exch cdr (pair2)
car (pair1) exch cdr (pair3)

cdr (pair1) exch car (pair2)
cdr (pair1) exch car (pair3)
cdr (pair1) coinc cdr (pair2)
cdr (pair1) coinc cdr (pair3)

car (pair2) coinc car (pair3)
cdr (pair2) coinc cdr (pair3)
car (pair2) exch cdr (pair3)
cdr (pair2) exch car (pair3)



comment:
black arrow: order relation
red arrow: coincoidence relation
green arrow: exchange relation
terms hd, tl are identical with themselves: id
(not all arrows are drawn in the diagram)

Diagramm 8 **Symmetric mediation table of pair$^{(2)}$**

pair $^{(2)}$	hd1	tl1	hd2	tl2
hd1	**id**	**ord**	**coinc**	**exch**
tl1	**ord**	**id**	**exch**	**coinc**
hd2	**coinc**	**exch**	**id**	**ord**
tl2	**exch**	**coinc**	**ord**	**id**

Short:

```
car (pair1) coinc car ( pair2) coinc car ( pair3)
cdr (pair1) coinc cdr ( pair2) coinc cdr ( pair3)

car (pair1) exch cdr ( pair2) coinc cdr ( pair3)
car (pair1) exch cdr ( pair2) exch cdr ( pair3)
cdr (pair1) exch car ( pair2) exch cdr ( pair3)
cdr (pair2) exch car ( pair2) exch car ( pair3)          ????
```

Abstract objects of typ list.

```
contexture
     names (=lists0)
        sorts
            element, list
        opn
           a, b, c, ..., z :               --> element
           nil             :               --> list
           cons            : element, list --> list
```

Poly-lists are implemented as mediated patterns of lists of ordered pairs belonging to different contextures. A pair is composed of a head and a tail. Poly-lists $list_i$ and $list_j$ are mediated by the proemial relations of order, exchange and coincidence of different loci. The head of each local list points to a local data item and the local tail points to the next local pair in the local list. This is a minimal condition for poly-lists.

A list is a 2-pointer object. A poly-list is a 2-pointer-2-"jumper" complexion or c-object.

Basic operations on poly-lists

repl (pairi) = paij

exch (headi) = (tailj)

exch (taili) = (headj)

Transjunctional Patterns

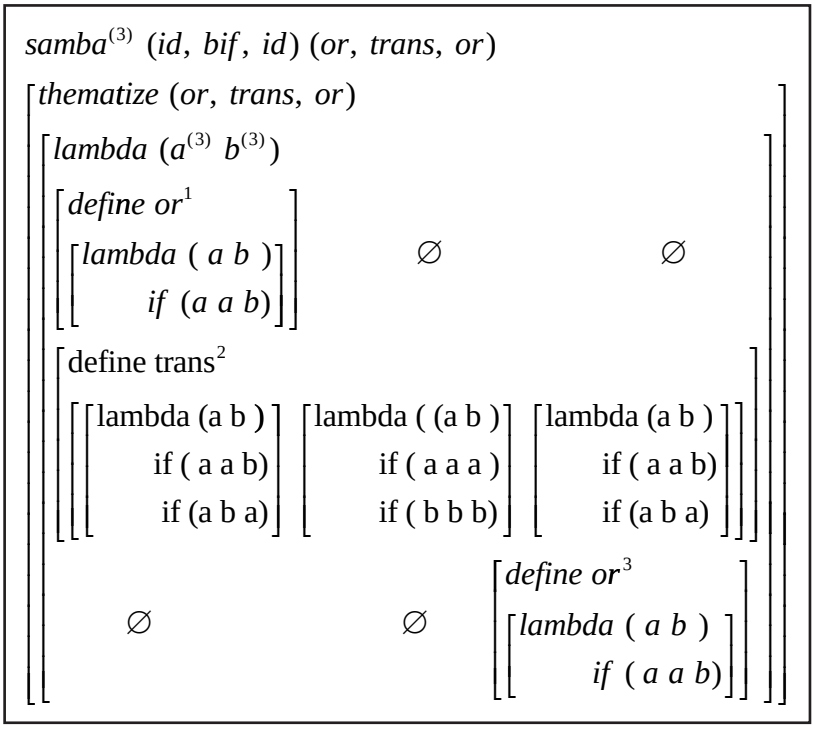
Rules for transjunctions in ConTeXtures are based on the polycontextural rules of transjunctions as defined by the analytical tableaux method. These rules are not easy to catch and in the literature they don't find any proper treatment. The tableaux rules introduced here goes a long way back into my own research and for some reasons I haven't published them – at least in this setting – until now. The different transjunction rules of ConTeXtures have to be understood as possible transcriptions from the logical thematization to a programming implementation. Transjunctions, together with junctions and negations plus the super-operators, could surely be used directly to define a poly-logical programming paradigm. This will be developed at another place. The job to do here is to transcribe or translate these polycontextural tableaux rules into functional lambda programs. That is, to make programs out of tableaux rules. As for all formalisms some implicit conventions of reading and understanding formulas, like tableaux, are involved. To produce the lambda programs in ConTeXture equivalent to the tableaux rules some conventions, not always easily to make explicit, are involved, too. Thus, there are different possibilities to implement the logical tableaux rules into lambda or samba programs. In other words, it's up to programmers to do it, if necessary, better. Only their use in a practical environment of programming will lead to some standard forms accepted by emerging conventions.

The tableaux rules of polycontextural logics are developed strictly in the spirit of Raymond Smullyan's pioneering work (1968). It has to be mentioned, that Smullyan's approach was the only one who helped me to understand and develop polycontextural logic. Neither Universal Algebra nor later Category Theory had been of much help on a strictly inventional and creational level.

19 General pattern for (id, bif, id)-modus of interaction

BOOLEAN TOPICS !!!

$$
\begin{array}{l}
samba^{(3)}\ (id,\ bif,\ id) \\
\left[\begin{array}{l}
thematize\ (junct,\ trans,\ junct) \\
\left[\begin{array}{l}
lambda\ (a^{(3)}\ b^{(3)}) \\
\left[\begin{array}{l}
\left[\begin{array}{l} define\ junction^{1} \\ \left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \quad if\ (cond - a\ b) \end{array}\right] \end{array}\right] \quad \varnothing \quad \varnothing \\
\left[\begin{array}{l} define\ trans^{2} \\ \left[\left[\begin{array}{l} lambda\ (a\ b\ ) \\ \quad if\ (\ cond\text{-}a\ b) \\ \quad if\ (cond\text{-}b\ a) \end{array}\right] \left[\begin{array}{l} lambda\ (\ (a\ b\ ) \\ \quad if\ (\ a\ a\ a\ ) \\ \quad if\ (\ b\ b\ b) \end{array}\right] \left[\begin{array}{l} lambda\ (a\ b\ ) \\ \quad if\ (\ cond\text{-}a\ b) \\ \quad if\ (cond\text{-}b\ a) \end{array}\right]\right] \end{array}\right] \\
\varnothing \quad \varnothing \quad \left[\begin{array}{l} define\ junction^{3} \\ \left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \quad if\ (\ cond\text{-}\ a\ b) \end{array}\right] \end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}
$$


samba(3) (id, bif, id) (or, trans, or)
thematize (or, trans, or)
lambda (a(3) b(3))
define or1
lambda (a b)
if (a a b)
∅
∅
define trans2
lambda (a b)
if (a a b)
if (a b a)
lambda ((a b)
if (a a a)
if (b b b)
lambda (a b)
if (a a b)
if (a b a)
∅
∅
define or3
lambda (a b)
if (a a b)

old

$$
\begin{array}{l}
samba^{(3)}\ (bif,\ id,\ id) \\
\left[\begin{array}{l}
thematize\ (\ trans,\ or,\ or\ ) \\
\left[\begin{array}{l}
lambda\ (a^{(3)}\ b^{(3)}) \\
\left[\begin{array}{l}
define\ trans^{1} \\
\left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ elect_{1} \\ \quad (\ if\ \ a\ a\ a) \\ \quad (\ if\ \ b\ b\ b) \end{array}\right]\left[\left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ elect_{2} \\ \quad (\ if\ \ b\ a\ b) \\ \quad (\ if\ \ b\ b\ a) \end{array}\right]\left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ elect_{3} \\ \quad (\ if\ \ b\ a\ b) \\ \quad (\ if\ \ b\ b\ a) \end{array}\right]\right]
\end{array}\right]
\end{array}\right] \\
\begin{array}{ccc}
\varnothing & \left[\begin{array}{l} define\ or^{2} \\ \left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \quad (\ if\ \ a\ a\ b) \end{array}\right] \\ elect_{1} \\ \quad (if\ \ b\ \ a\ \ b\ ) \end{array}\right] & \varnothing \\
\varnothing & \varnothing & \left[\begin{array}{l} define\ or^{3} \\ \left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \quad (\ if\ \ a\ a\ b) \end{array}\right] \\ elect_{1} \\ \quad (if\ \ b\ \ a\ \ b\ ) \end{array}\right]
\end{array}
\end{array}\right]
\end{array}
$$

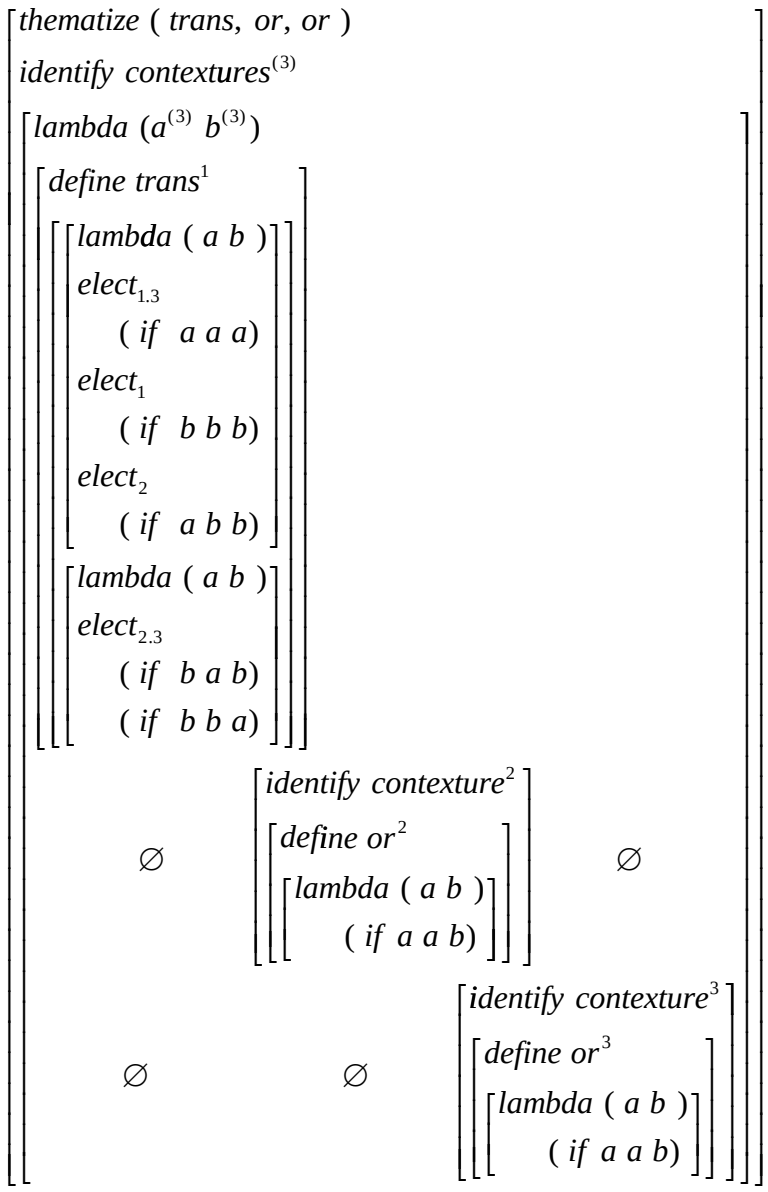
not the real thing ??

$(trans,\ or,\ or)$	$O1$	$O2$	$O3$
$M1$	S_1	$\varnothing$	$\varnothing$
$M2$	S_2	S_2	$\varnothing$
$M3$	S_3	$\varnothing$	S_3

$$samba^{(3)}\ (bif,\ id,\ id)$$

$$\left[\begin{array}{l} thematize\ (\ trans,\ or,\ or\ ) \\ \left[\begin{array}{l} lambda\ (a^{(3)}\ b^{(3)}) \\ \left[\begin{array}{l} define\ trans^{1} \\ \left[\left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \left[\begin{array}{l} elect_{1} \\ \quad (\ if\ \ a\ a\ a) \\ \quad (\ if\ \ b\ b\ b) \end{array}\right] \end{array}\right]\right. \\ \left[\left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \left[\begin{array}{l} elect_{2} \\ \quad (\ if\ \ b\ a\ b) \\ \quad (\ if\ \ b\ b\ a) \end{array}\right] \end{array}\right]\right. \\ \left.\left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \left[\begin{array}{l} elect_{3} \\ \quad (\ if\ \ b\ a\ b) \\ \quad (\ if\ \ b\ b\ a) \end{array}\right] \end{array}\right]\right] \end{array}\right] \\ \qquad \varnothing \qquad \left[\begin{array}{l} define\ or^{2} \\ \left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \quad (\ if\ a\ a\ b) \end{array}\right] \\ elect_{1} \\ \quad (if\ \ b\ a\ b\ ) \end{array}\right] \qquad \varnothing \\ \qquad \varnothing \qquad\qquad \varnothing \qquad \left[\begin{array}{l} define\ or^{3} \\ \left[\begin{array}{l} lambda\ (\ a\ b\ ) \\ \quad (\ if\ a\ a\ b) \end{array}\right] \\ elect_{1} \\ \quad (if\ \ b\ a\ b\ ) \end{array}\right] \end{array}\right] \end{array}\right]$$

confusing ??


thematize (trans, or, or)
identify contextures(3)
lambda (a(3) b(3))
define trans1
lambda (a b)
elect1.3
(if a a a)
elect1
(if b b b)
elect2
(if a b b)
lambda (a b)
elect2.3
(if b a b)
(if b b a)
∅
identify contexture2
define or2
lambda (a b)
(if a a b)
∅
∅
∅
identify contexture3
define or3
lambda (a b)
(if a a b)

$$
\begin{array}{l}
samba^{(3)}\ (\ bif,\ id,\ id\ ) \\
\left[\begin{array}{l}
thematize\ (\ trans,\ or,\ or\ ) \\
identify\ contextures^{(3)} \\
\left[\begin{array}{l}
lambda\ (a^{(3)}\ b^{(3)}) \\
\left[\begin{array}{l}
define\ trans^{1} \\
\left[\left[\left[\left[\left[\begin{array}{l}
elect_{1.3} \\
\quad (\ if\ \ a\ a\ a) \\
elect_{1} \\
\quad (\ if\ \ b\ b\ b) \\
elect_{2} \\
\quad (\ if\ \ a\ b\ b) \\
elect_{2.3} \\
\quad (\ if\ \ b\ a\ b) \\
\quad (\ if\ \ b\ b\ a)
\end{array}\right]\right]\right]\right]\right]
\end{array}\right]
\left[\left[\begin{array}{l}
identify\ contexture^{2} \\
\left[\begin{array}{l}
define\ or^{2} \\
\left[\left[\begin{array}{l}
lambda\ (\ a\ b\ ) \\
\quad (\ if\ \ a\ a\ b)
\end{array}\right]\right]
\end{array}\right]
\end{array}\right]\right]
\left[\left[\begin{array}{l}
identify\ contexture^{3} \\
\left[\begin{array}{l}
define\ or^{3} \\
\left[\left[\begin{array}{l}
lambda\ (\ a\ b\ ) \\
\quad (\ if\ \ a\ a\ b)
\end{array}\right]\right]
\end{array}\right]
\end{array}\right]\right]
\end{array}\right]
\end{array}
$$

$$
\begin{array}{l}
samba^{(3)}(\ bif,\ id,\ id\ ) \\
\left[\begin{array}{l}
thematize\ (\ trans,\ or,\ or\ ) \\
identify\ contextures^{(3)} \\
\left[\begin{array}{l}
lambda\ (\ a^{(3)}\ b^{(3)}) \\
\left[\begin{array}{l}
define\ trans^{1} \\
\left[\left[\left[\left[\left[\begin{array}{l}
(\ elect_{1.3}\ a\ a\ a) \\
(\ elect_{1}\ \ b\ b\ b) \\
(\ elect_{2}\ \ a\ b\ b) \\
(\ elect_{2.3}\ b\ a\ b) \\
(\ elect_{2.3}\ b\ b\ a)
\end{array}\right]\right]\right]\right]\right]
\end{array}\right]
\left[\left[\begin{array}{l}
identify\ contexture^{2} \\
\left[\begin{array}{l}
define\ or^{2} \\
\left[\left[\begin{array}{l}
lambda\ (\ a\ b\ ) \\
\quad (\ if\ \ a\ a\ b)
\end{array}\right]\right]
\end{array}\right]
\end{array}\right]\right]
\left[\left[\begin{array}{l}
identify\ contexture^{3} \\
\left[\begin{array}{l}
define\ or^{3} \\
\left[\left[\begin{array}{l}
lambda\ (\ a\ b\ ) \\
\quad (\ if\ \ a\ a\ b)
\end{array}\right]\right]
\end{array}\right]
\end{array}\right]\right]
\end{array}\right]
\end{array}
$$

```
trans1 ( a b )(3):
( a a )1 --> a1, a3,    ( a a )1 --> ( sel1 a ), ( elect3 a )
( b b )1 --> b1, a2
( a b )1 --> b2, b3
( b a )1 --> b2, b3
```

samba ((id, bif2, id3), ARS, 3, (or, trans, or))

First step intro

(id1 (define or1 (lambda (a b)1
if1 (a a b))))

(bif2 (define trans2 (lambda ((a b)2, (a b)1, (a b)3)
if2 (a a a)
if2 (b b b)
simul
if1 (a a b)
if1 (a b a)
simul
if3 (a a b)
if3 (a b a)

(id3 (define or3 (lambda (a b)3
if3 (a a b))))

(or, trans, or): ARS1 * ARS 2* ARS3 ——> ARS1 * (ARS2, ARS1, ARS3) * ARS3

Explicite: (or, trans, or): (ARS1 , . , .) * (ARS1 , . , .) * (ARS1 , . , .) ——>

(ARS1 , . , .) * (ARS1, ARS2 , ARS3) * (., . , ARS3)

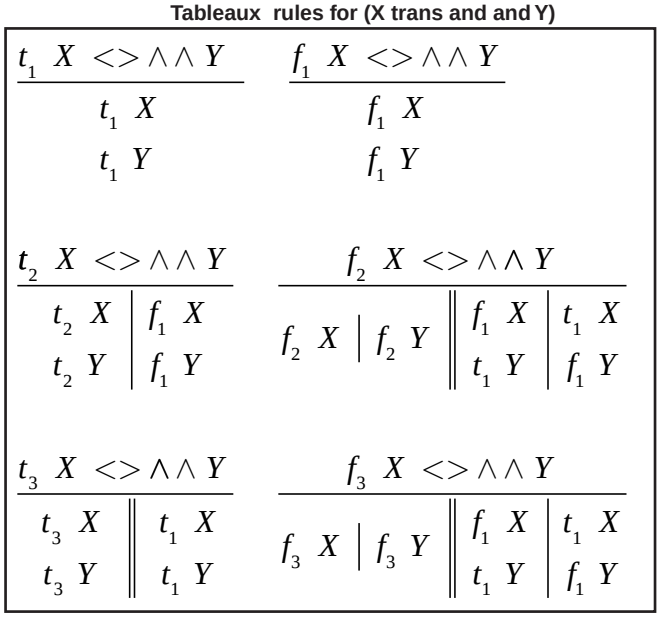
`"*": mediation from samba`

(or, trans, or) corresponds to the 3-contextural binary logical function of (disjunction, transjunction, disjunction).

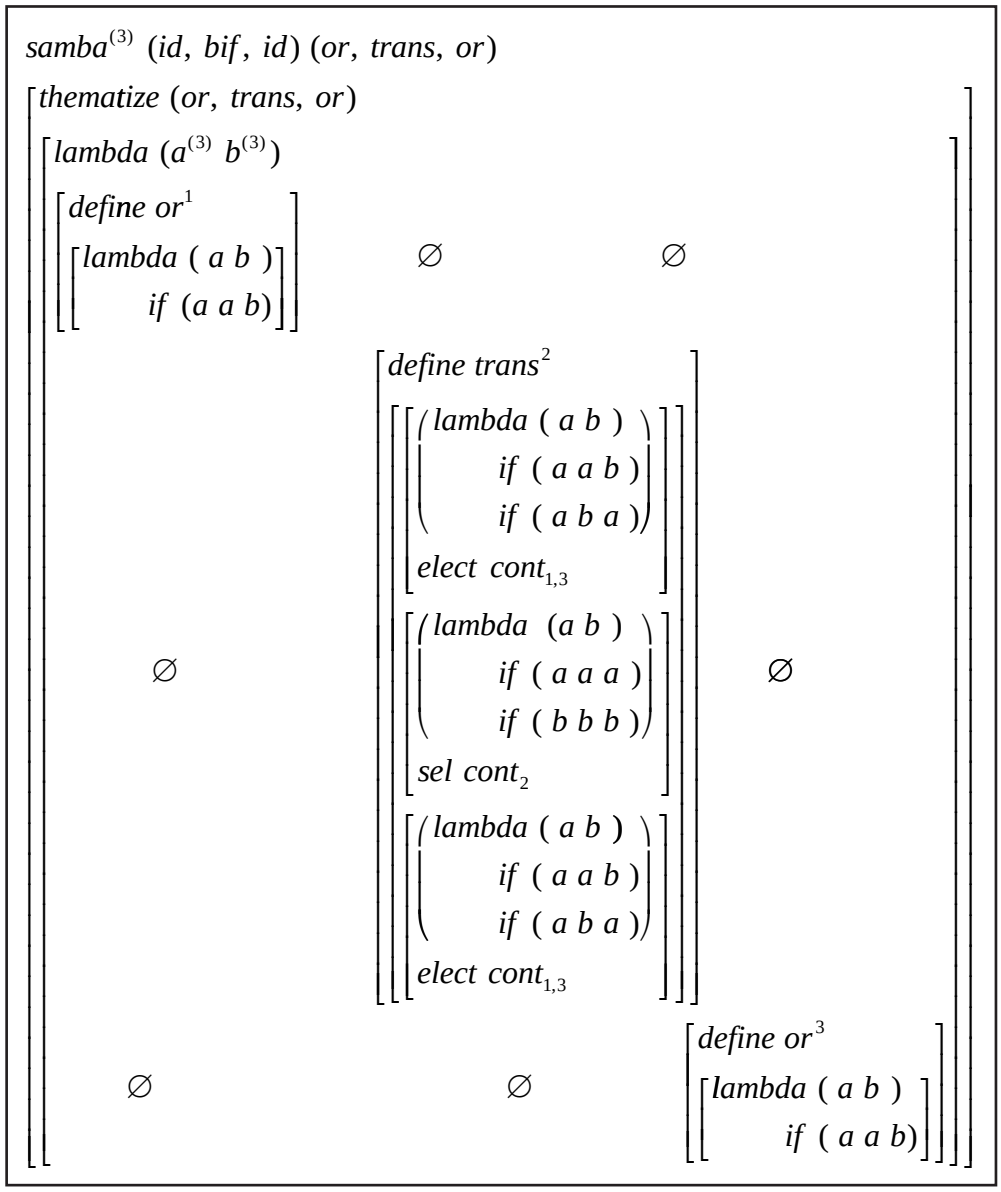
19.1 Tableaux Method for Transjunctions in $G^{(3)}$

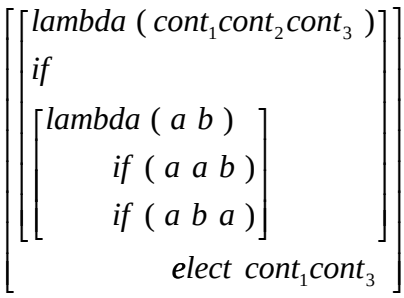
Rules for transjunctions in ConTeXtures are based on the polycontextural rules of transjunctions as defined by the analytical tableaux method. These rules are not easy to crasp and in the literature they don't find any proper treatment. The tableaux rules introduced here goes a long way back into my own research and for some reasons I haven't published them – in this setting – until now. The different transjunction rules of ConTeXtures have to be understood as possible transcriptions from the logical thematization to a programming implementation. Transjunctions, together with junctions and negations plus the super-operators, could surely be used directly to define a poly-logical programming paradigm. This will be developed at another place. The job to do here is to transcribe or translate these polycontextural tableaux rules into functional lambda programs. That is, to make programs out of tableaux rules. As for all formalisms some implicit conventions of reading and understanding formulas, like tableaux, are involved. To produce the lambda programs in ConTeXture equivalent to the tableaux rules some conventions, not always easily to make explicit, are involved, too. Thus, there are different possibilities to implement the logical tableaux rules into lambda or samba programs. In other words, it's up to programmers to do it, if necessary, better. Only there use in a practical environment of programming will lead to some standard forms accepted by emerging conventions. The tableaux rules of polycontextural logics are developed strictly in the spirit of Raymond Smullyan's pioneering work (1968). It has to be mentioned, that Smullyan's approach was the only one who helped me to understand and develop polycontextural logic. Neither Universal Algebra nor later Category Theory had been of much help on a strictly inventional and creational level. Cf. http://www.thinkartlab.com/pkl/lola/PolyLogics.pdf

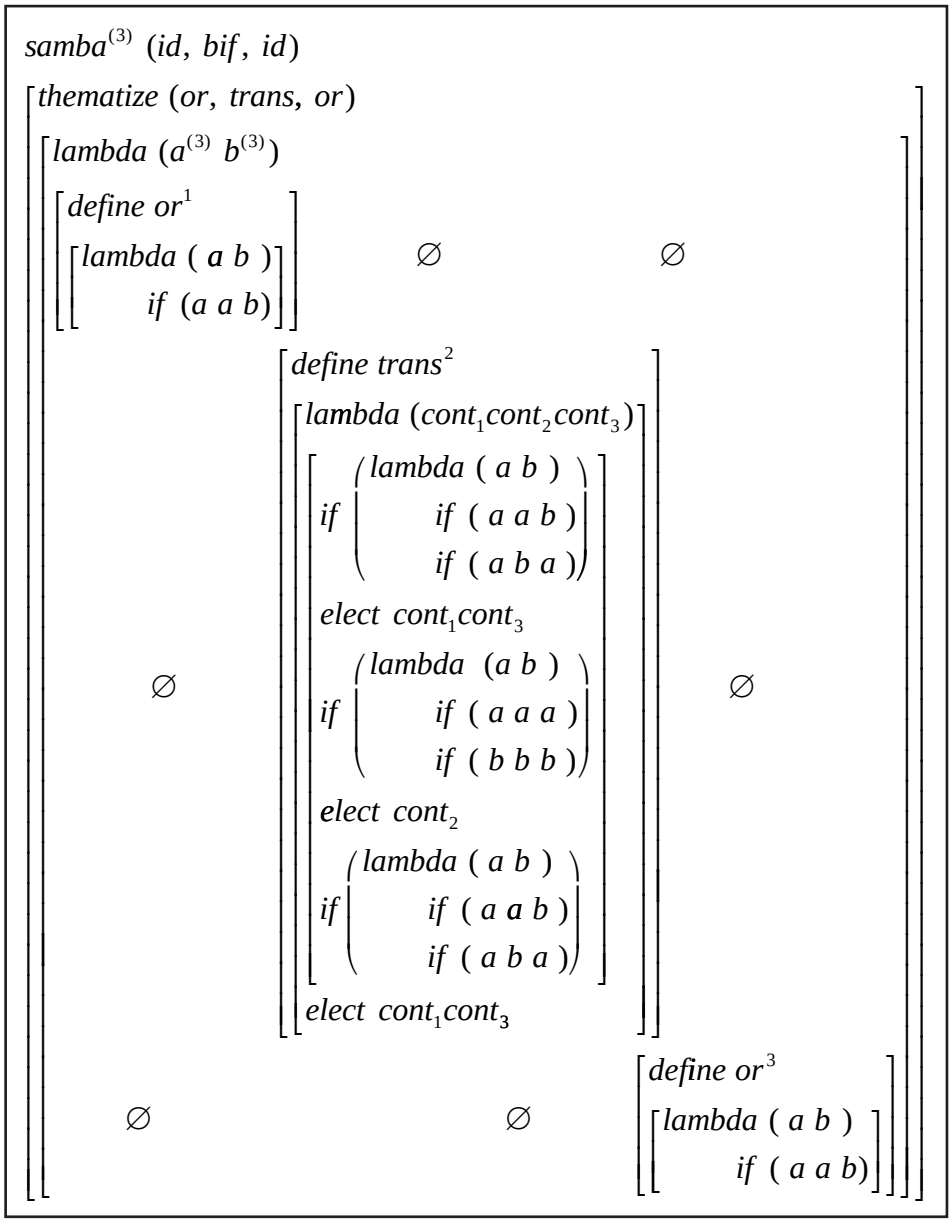
Diagramm 9 **Tableaux rules for (X trans and and Y)**



These tableau rules for the logical function (trans, and, and) gives the full information about the definition and the interaction of the local parts of the global function.


samba(3) (id, bif, id) (or, trans, or)
thematize (or, trans, or)
lambda (a(3) b(3))
define or1
lambda (a b)
if (a a b)
define trans2
lambda (a b)
if (a a b)
if (a b a)
elect cont1,3
lambda (a b)
if (a a a)
if (b b b)
sel cont2
lambda (a b)
if (a a b)
if (a b a)
elect cont1,3
define or3
lambda (a b)
if (a a b)


lambda (cont1cont2cont3)
if
lambda (a b)
if (a a b)
if (a b a)
elect cont1cont3



Contextures as names of the lambda abstraction. Contextures can be named and are, at once, the locus or frame of names and the naming process. This is introduced from the local viewpoint of contexture2.

And conditional if.

Instead of: lambda (contextures) samba (contextures) ??

19.2 General pattern for (bif, bif, bif)-modus of interaction

Diagramm 10 **Logical Transjunctions**

```
samba(3) ( bif, bif, bif )
[thematize ( transj, transj, transj )
 [lambda (a(3) b(3))
  [define trans1
   [[[lambda ( a b )
      ( if cond-a b )
      ( if cond-b a )]
     [lambda ( a b )
      ( if a a a )
      ( if b b b )]
     [lambda ( a b )
      ( if cond-a b )
      ( if cond-b a )]]
  [define trans2
   [[[lambda ( a b )
      ( if cond-a b )
      ( if cond-b a )]
     [lambda ( a b )
      ( if a a a )
      ( if b b b )]
     [lambda ( a b )
      ( if cond-a b )
      ( if cond-b a )]]
  [define trans3
   [[[lambda ( a b )
      ( if cond-a b )
      ( if cond-b a )]
     [lambda ( a b )
      ( if a a a )
      ( if b b b )]
     [lambda ( a b )
      ( if cond-a b )
      ( if cond-b a )]]]]]]]
```

This is the scheme for the so called total transjunction in polycontextural logics.

Depending on the conditions "cond" the different total transjunctions are defined. They are traditionally called conjunctive, disjunctive, implicative, replicative, etc. transjunctions. The distinction between disjunctive and conjunctive transjunctions comes into play for m>=4 (Kaehr, Pfalzgraf).

It is well known that total transjunctions of any complexity can have the properties of quasi-algebras and have a nice representation as Latin squares (Kaehr 1976).

Formula is wrong: (a a a) per subsystem !!!

(trans, trans, trans):

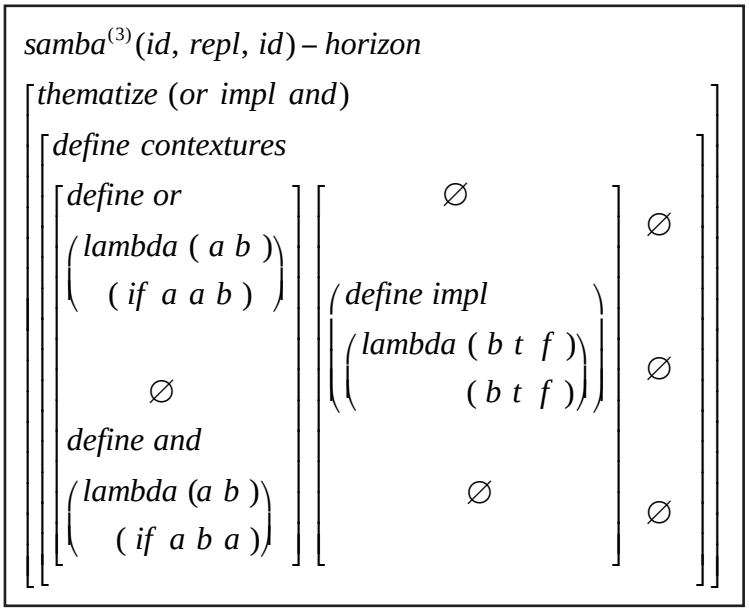
trans[1]: cond–a b in O2 and O3
total : if a a b , if a b a :: (1 3 3 2)
implicative: if b a b , if a b a :: (1 3 1 2)
replicative: if a a b , if b a b :: (1 1 3 2)

trans[2]: cond–a b in O1 and O3
total : if a a b , if a b a :: (2 1 1 3)
implicative: if b a b , if a b a :: (2 2 1 3)
replicative: if a a b , if b a b :: (2 1 2 3)

trans[3]: cond–a b in O1 and O2
total : if a a b , if a b a :: (1 2 2 3)
implicative: if b a b , if a b a :: (1 2 1 3)
replicative: if a a b , if b a b :: (1 2 1 3)

Tableaux logic representation of the total tranjunction (trans, trans, trans)

19.3 General pattern for (id, id, red)-modus of interaction



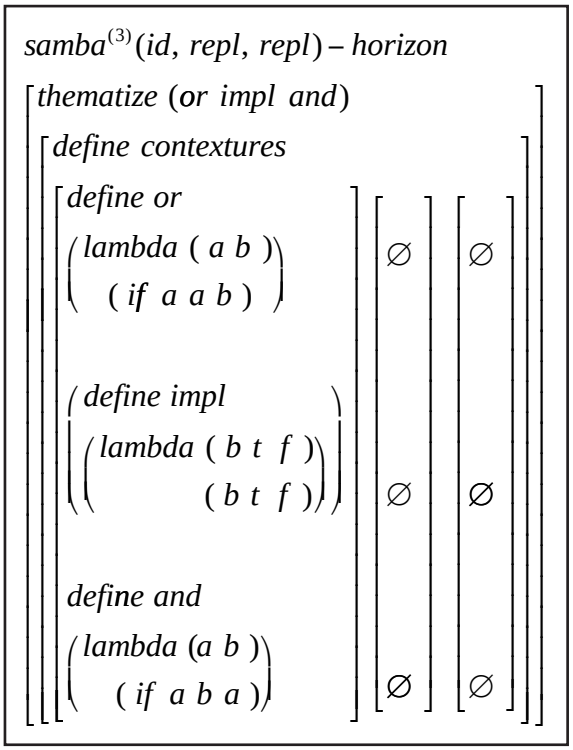
Pattern: [S101, S020, S000]
Interactivity: [(id, empt, repl) (emt, id, emt) (emt emt empt)
Constellation: [(or, empt, and) (empt, impl, empt) (emt emt emt)]
Topic: Binary Boolean

NOT REPLICATION!!! REDUCTION!!!

(or impl and) as (id, id, red)
S1 —> S1 : id1
S2 —> S2 : id2
S3 —> S1 : red31, reduction S3 to S1

The definition at O1M1 is complete: head plus body. It is not modeling a reflectional, that is, replicative situation but a reduction from O3M3 to O1M3. Therefore the bracket presentation is correct.

Tableau for (or impl and) (S121)



Pattern: [S111, S000, S000]
Interactivity: [(id, repl, repl) (empty) (empty)]
Constellation: [(or impl and) (empty) (empty)]
Topic: Binary Boolean

(or impl and) as (id red red): S123 —> S111

!!! not repl but red !!!?

Tableau for (or impl and) (S111)

Permutational Patterns

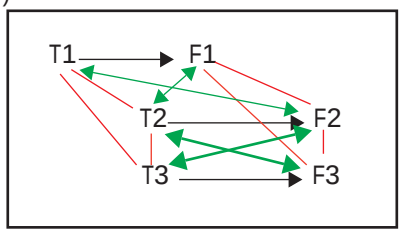
20 General pattern for (id, perm, perm)-modus of interaction

Matrix (id, perm, perm)

20.1 System changes with Negations

To understand the mechanism of system changes within negations it is necessary to have a new look at our Booleans. They are distributed and mediated pairs of truth values.

Boolean truth values(3)



It is easy to see that a negation in one system is producing permutations or transversions in other systems.

N1: S123 —> S132

With N1 (T1,T2,T3) => (F1, T3,T2) and N1 (F1, F2, F3) => (T1, F3, F2)

But this is only half the cake. Without considering the reflectional complex architecture of the system, this is perfectly correct and a lot of work has been done under this presumption. One part of that presumption is the neglection of the parallelism of the formal processes. This point is not only highly visible for transjunctional constructions but comes into focus also for strictly junctional and negational relations and procedures.

From the viewpoint of a reflectional and processual understanding we have to make a distinction between the values of a systems an the same values from another system. They are not identical because they have another "history" , that is, their formulas start with a different root, they can not been be taken as identical but as the same. Their sameness demands for a special locus to place them otherwise they will collide with the genuine values or objects of their guest system.

System change for negation non1 in a reflectional architecture:

```
S123 --> S132:  (S100, S020, S003) --> (S100, S003, S020)
```

20.2 Boolean topics: Negations

Negations non1

samba (define **not1** (lambda(3) (sel1)
id (sel (false1 true1))
perm (false2 true2)
perm (false3 true3)

samba(3) (define **not1**(lambda1 (sel1)
(id perm perm (sel1 (true(3) false(3)))))

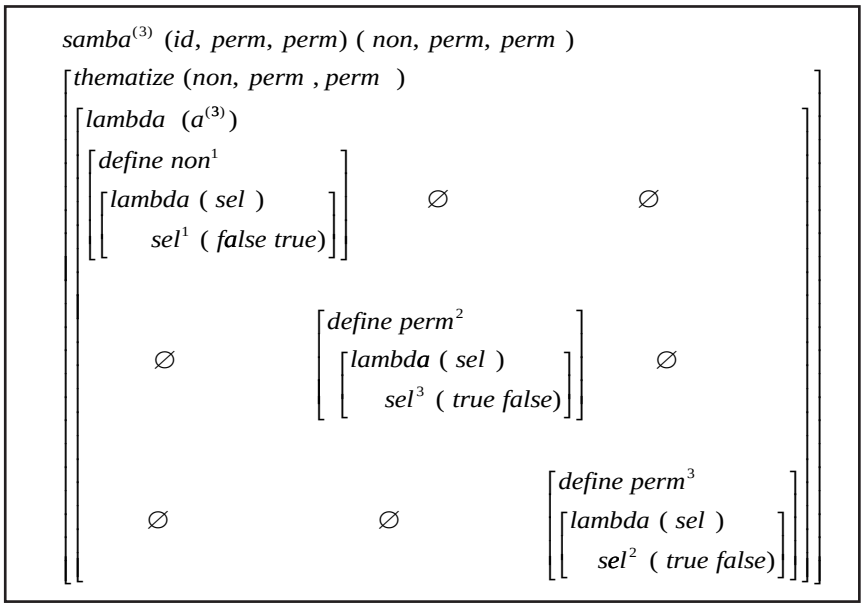
bdisp! samba (define **not1** (lambda(3) (sel1):
bdisp! id (sel (false1 true1) —> bdisp! not1 (true1 false1) —> (false1 true1)
bdisp! perm (false2 true2) —> bdisp! perm (true2 false2) —> (true3 false3)
bdisp! perm (false3 true3) —> bdisp! perm (true3 fals3) —> (true2 false2)

```
(bdisp! (not1 true(3)))
                          --> (false1, true3, true2)
(bdisp! (not1 false(3)))
                          --> (true1, false3, false2)
```

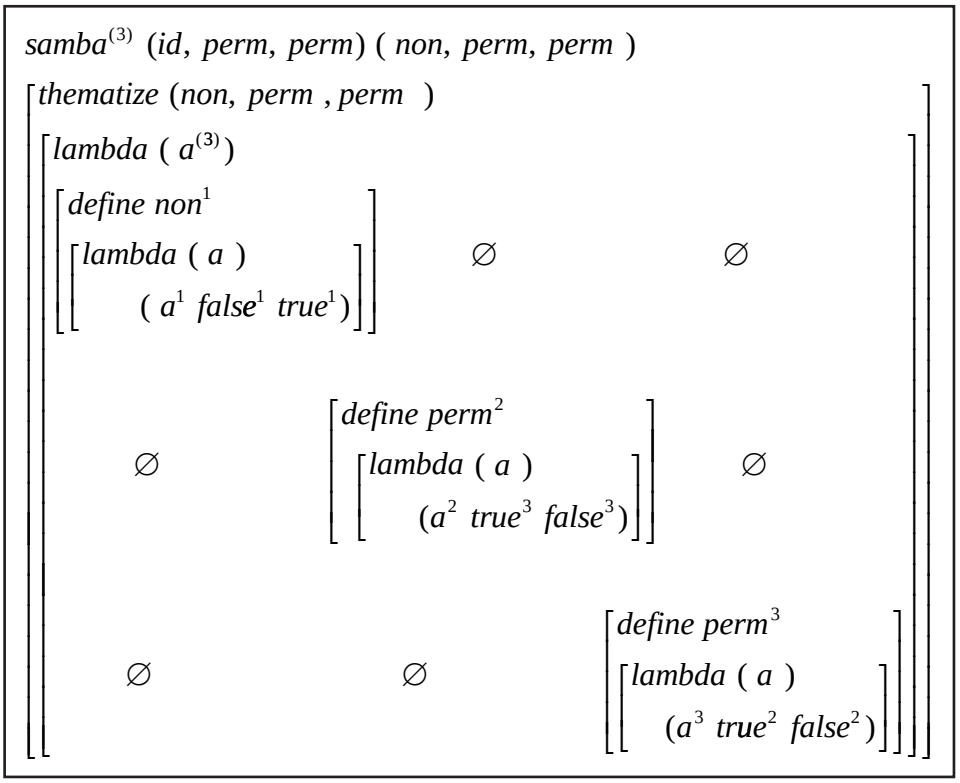
samba ((id, perm) ARS, 3, not1)

(id1 (define not1 (lambda (sel) (sel only in S1 and not also in S2,3!!)
sel (false1 true1))))
(perm2 (define not1 (lambda (sel) (perm2 (define not1 (lambda (sel2)
sel (false3 true3)))) sel2 (false2, true2)))) –> S3
(perm2 (define not1 (lambda (sel) (perm2 (define not1 (lambda (sel)
sel (false2 true2)))) sel (false3 true)))

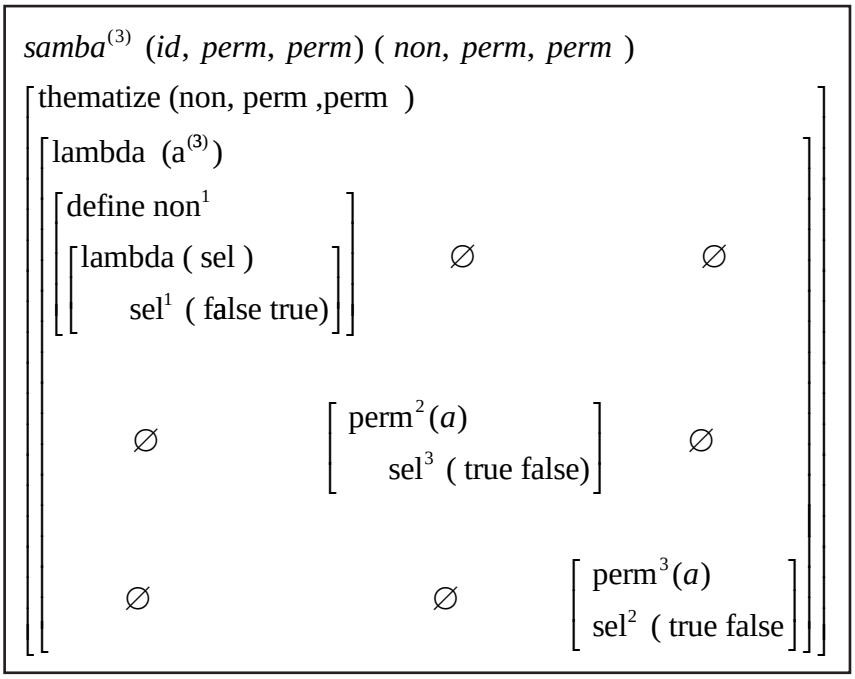
link: Boolean (id, id, id)



Another definition



"If a3 then (true2, false2)"



If a1 take (false, true)1 (= negation as inversion: inv(true, false)1)
If a2 take or go to (true, false)3
If a3 move to (true, false)2

define **perm2** ($a^{(3)}$)
lambda (a_2)
a_3

Negation non2

samba ((id, perm) ARS, 3, not2)

```
( perm2  (define not2 ( lambda ( sel )
                sel ( false3 true3 ))))
( id2 (define not2 ( lambda ( sel )
                sel ( false2 true2 ))))
( perm2 (define not2 ( lambda ( sel )
                sel ( false1 true1 ))))
```

define not3 (lambda **(not1)**
lambda (**not2**))
(not1 (not2 (not1)))
define not3 lambda (not1 not2)
compose (not1 not2 not1)

20.3 Distribution of Boolean lattices

Additionally to the strict id-modus of distribution the laws of negations has be introduced to define Boolean Lattices.

Distributing distributive latices

Identity: X * X ==X

Commutativity: X * Y == Y * X, *, + = {and, or}

Associativity: X * (Y * Z) == (X * Y) * Z

Absorbtion: X * (X + Y) == X,

Distributivity: X * (Y + Z) == (X * Y) + (X * Z)

Negations

Idempotence: noni (nonj (X(3)) = X(3), i=j=1,2

Cyclicity: non1 (non2 (non1 (X(3)))) = non2 (non1 (non2 (X(3))))

Definitions: non3 (X(3)) = non1 (non2 (non1 (X(3))
= non2 (non1 (non2 (X(3))

DeMorgan laws for 3-Boolean lattices

X***Y = non3 (non3 (X(3)) +++ non3 (X(3))
X *+* Y = non3 (non3 (X(3)) +*+ non3 (X(3))

$L_{distr} = (com, assoc, abs, distr)$

$L^{(3)}_{distr} : (L^{1}_{distr}, L^{2}_{distr}, L^{3}_{distr}) \longrightarrow L^{(3)}_{distr}$

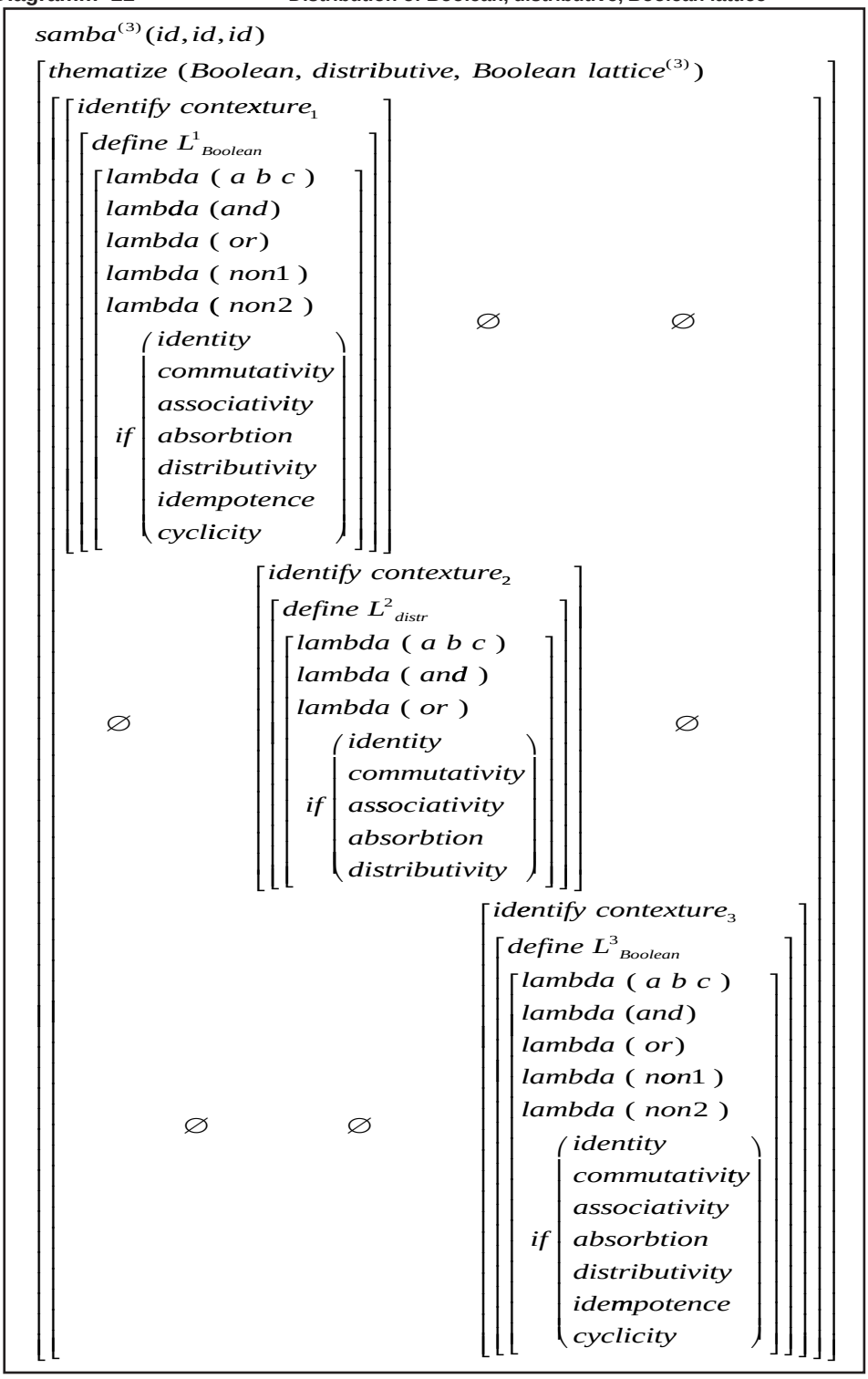
Permutation and identity

A distribution of Boolean lattices over 3 contextures belongs to the template [(S100), (S020), (S003)] and the pattern [id, id, id].

But the negations are using the superoperator perm. Only because of the symmetric definition of the DeMorgan laws this permutation is transformed to the identity of

id: perm (perm (id))) = id

Diagramm 11 **Distribution of Boolean, distributive, Boolean lattice**



21 Application of extended logical abstractions

local negations

```
(bdisp! (truei))
                    --> truei

(bdisp! (falsei))
                    --> false1
(bdisp! (true(3)))
                    --> (true1 true2 true3)

(bdisp! (fals(3)))
                    --> ( false1 false2 fals3)

(bdisp! (notitruei))
                           --> falsei

(bdisp! (notifalsei))
                           --> truei,       i= 1,2
```

mediated negations

```
(bdisp! (not1 true(3)))
                           --> (false1, true3, true2)

(bdisp! (not1 false(3)))
                           --> (true1, false3, false2)

(bdisp! (not2 true(3)))
                          --> ( true3, false2, true1)

(bdisp! (not2 false(3)))
                           --> ( false3, true2, false1)
```

Junctions, mono- and polyform

```
(bdisp! (and(3) true(3) true(3)))
                           --> (true1, true2, true3)
(bdisp! (and(3) false(3) false(3)))
                           --> (false1, false2, false3)
(bdisp! (and(3) true(3) false(3)))
                           --> (false1, false2, false3)
(bdisp! (and(3) false(3) true(3)))
                           --> (false1, false2, false3)

(bdisp! (or(3) true(3) true(3)))
                           --> (true1, true2, true3)
(bdisp! (or(3) false(3) false(3)))
                           --> (false1, false2, false3)
(bdisp! (or(3) true(3) false(3)))
                           --> (true1, true2, true3)
(bdisp! (or(3) false(3) true(3)))
                           --> (true1, true2, true3)



(bdisp! ((and,or,or) true(3) true(3)))
                           --> (true1, true2, true3)
```

```
(bdisp! ((and,or,or) false(3) false(3)))
                          --> (false1, false2, false3)

(bdisp! ((and,or,or) true(3) false(3)))
                          --> (false1, true2, true3)

(bdisp! ((and,or,or) false(3) true(3)))
                          --> (false1, true2, true3)
```

Junctions plus transjunctions

```
(bdisp! ((trans,or,and) true(3) true(3)))
                          --> (true1, true2, true3)

(bdisp! ((trans,or,and) false(3) false(3)))
                          --> (false1, false2, false3)

(bdisp! ((trans,or,and) true(3) false(3)))
                         --> ( ( empt1 , false2, false3), true2, false3)

(bdisp! ((trans,or,and) false(3) true(3)))
                          --> (( empt1 , false2, false3), true2, false3)
```

22 Arithmeltical topics

samba ((bif1, id, id), ARS, 3, zero(3))

```
    ( define zero1 (lambda ( f )
                    ( lambda  ( x )
                       x ))
     ( define zero2 (lambda ( f )
                    ( lambda  ( x )
                       x ))
simul
     ( define zero1 (lambda ( f )
                    ( lambda  ( x )
                       x ))
     ( define zero3 (lambda ( f )
                    ( lambda  ( x )
                       x ))

ndisp! : ((0, . , . ), (0, 0, . ), ( . , . , 0))
```

samba (id1, bif2, id3, ARS, 3, compose)

samba1 compose
samba2 compose . simul. samba1 compose
samba3 compose

samba (bif1, ARS, 3, succ)

```
 ( ID1 ( define  succ1    ( lambda (n )
                      ( lambda ( f )
                         ( compose1  f ( n f )))))))
 ( BIF1 ( define  succ2   ( lambda (n )
                      ( lambda ( f )
                         ( compose2  f ( n f ))))))
simul
                     ( lambda (n )
                       ( lambda ( f )
                          ( compose1  f ( n f ))))))
  ( ID3 ( define  succ3    ( lambda (n )
                      ( lambda ( f )
                         ( compose  f ( n f ))))))
```

bif2, red1, red2, perm1, perm2 : trivial.

bif1 (succ(3) succ1
succ2 simul succ1
succ3)

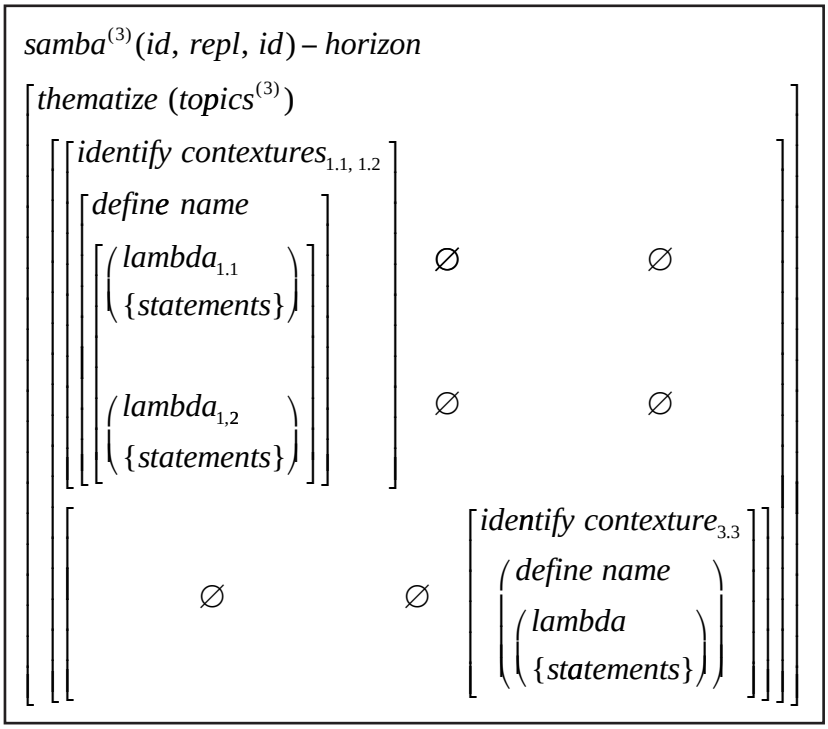
samba ((trans, id, id), ARS, 3, (transpose compose compose))

```
 ( perm ( define transpose1   ( lambda ( f  )1
                                   transition (f ))       general
( id ( define compose2   ( lambda ( f g )
```

```
                                    ( lambda ( x )
                                        (f (g x ))))
( id  ( define compose3   ( lambda ( f g )
                                    ( lambda ( x )
                                        (f (g x ))))
```

samba (ARS, 3, neibgh)

```
 ( perm ( define  neibgh1 ( lambda (n )              numeric
                        ( lambda ( f )
                            ( transpose1  ( n f )))))))
```



samba(3)(id, repl, id) – horizon
thematize (topics(3))
identify contextures1.1, 1.2
define name
lambda1.1
{statements}
lambda1,2
{statements}
∅
∅
∅
∅
∅
∅
identify contexture3.3
define name
lambda
{statements}

Why not jump from Y to Why?

24 Y-Operator

```
( define Y
   (lambda (f)
      ((lambda (x)
          ( f (x x )))
       (lambda ( x )
          ( f ( x x ))))))
```

$$\left[\begin{array}{l} define\ Y \\ \left[\begin{array}{l} lambda\ (f) \\ \left[\begin{array}{l} \left[\begin{array}{l} lambda\ (x) \\ \qquad\qquad (f\ (x\ x\ )) \end{array}\right] \\ \left[\begin{array}{l} lambda\ (x) \\ \qquad\qquad (f\ (x\ x\ )) \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

24.1 Distributed Y-Operators

$$samba^{(3)}(id, id, id) - horizon$$

$$\left[\left[\begin{array}{l} thematize\ (\ Y,\ Y,\ Y\ ) \\ \left[\begin{array}{l} lambda\ (\ f^1\ f^2\ f^3\ ) \\ \left[\begin{array}{l} define\ Y \\ \left[\begin{array}{l} lambda\ (f) \\ \left[\left(\begin{array}{l} lambda\ (x) \\ \qquad (f\ (x\ x\ )) \end{array}\right) \left(\begin{array}{l} lambda\ (x) \\ \qquad (f\ (x\ x\ )) \end{array}\right)\right] \end{array}\right] \end{array}\right] \\ \left[\begin{array}{l} define\ Y \\ \left[\begin{array}{l} lambda\ (f) \\ \left[\left(\begin{array}{l} lambda\ (x) \\ \qquad (f\ (x\ x\ )) \end{array}\right) \left(\begin{array}{l} lambda\ (x) \\ \qquad (f\ (x\ x\ )) \end{array}\right)\right] \end{array}\right] \end{array}\right] \\ \left[\begin{array}{l} define\ Y \\ \left[\begin{array}{l} lambda\ (f) \\ \left[\left(\begin{array}{l} lambda\ (x) \\ \qquad (f\ (x\ x\ )) \end{array}\right) \left(\begin{array}{l} lambda\ (x) \\ \qquad (f\ (x\ x\ )) \end{array}\right)\right] \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]\right]$$

Pattern: $[(S_{100}), (S_{020}), (S_{003})]$

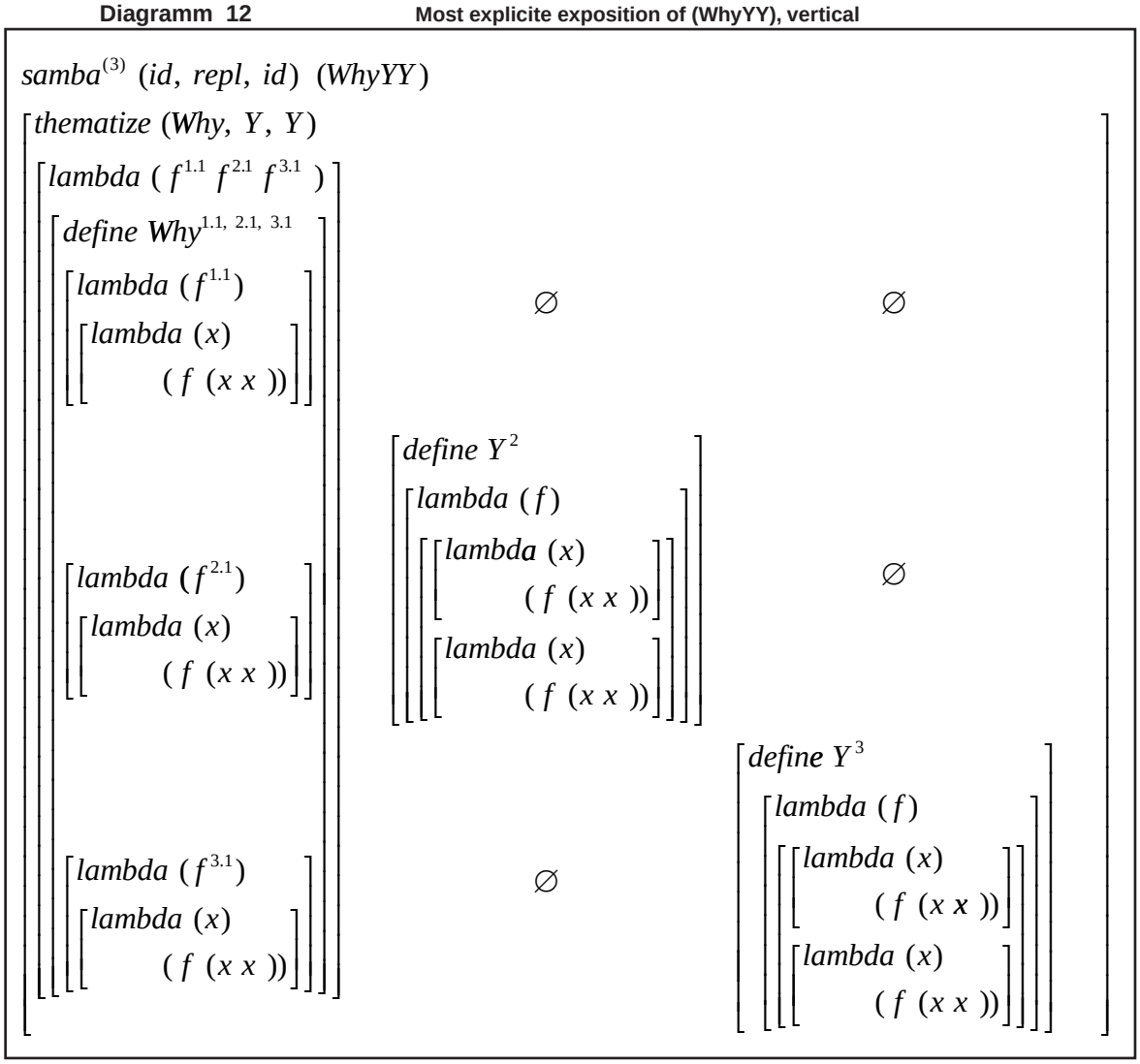
24.2 Why-operator

$$
\begin{array}{l}
samba^{(3)}(id,\ repl,\ id) - horizon \\
\left[\begin{array}{l}
thematize\ (\ Why^{1}\varnothing\varnothing\ ) \\
\left[\begin{array}{l}
lambda\ (\ f^{1.1}\ f^{1.2}\ f^{1.3}\ ) \\
\left[\begin{array}{l}
identify\ contexture^{1} \\
\left[\begin{array}{l}
define\ Why^{1} \\
\left[\begin{array}{l}
lambda\ (\ f^{1.1}\ ) \\
\left(\begin{array}{l} lambda\ (x) \\ \qquad (\ f\ (x\ x\ ))\end{array}\right)
\end{array}\right] \\
\left[\begin{array}{l}
lambda\ (\ f^{1.2}\ ) \\
\left(\begin{array}{l} lambda\ (x) \\ \qquad (\ f\ (x\ x\ ))\end{array}\right)
\end{array}\right] \\
\left[\begin{array}{l}
lambda\ (\ f^{1.3}\ ) \\
\left(\begin{array}{l} lambda\ (x) \\ \qquad (\ f\ (x\ x\ ))\end{array}\right)
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}
$$

Sup-ops (id, repl, id) defines the pattern: [(S100), (S120), (S103)]

24.3 Combined Y-Why-operators

Diagramm 12 **Most explicite exposition of (WhyYY), vertical**



Pattern
global: *Why* in systen S1 replicated over S2 and S3: [S100], [S120], [S103]
local: *Y* in systen S2 and S3 : [S020], [S003]

Diagramm 13 **Distribution pattern for WhyYY**

$$\begin{bmatrix} [[S11][S12][S13]] \\ [[S21][S22][S23]] \\ [[S31][S32][S33]] \end{bmatrix}$$

Distribution of Why and Y over 3 loci: (Why1, Y2, Y3)

The operator Why is defined as a reflectional distribution over the loci S11, S21 and S31. The Why-operator is not a single identity, but distributed and mediated over 3 different neighbor-systems which are reflecting it. This is ruled not by identity, but by sameness. The defining part is not simply iterated at the same locus, as it has to be done for the Y-operator, but reflected or alterated over 2 different mirroring loci. The third locus S31 is the place-holder for a comparison between the results of the 2 loci.

The operator Y-operators Y2 and Y3 are allo-referentially defined at their own loci, S22 and S33, that is, without reflection onto neighbor systems. They are what they are: Y-operators in their singular identity realized at different places.

F. Poly-Topics

25 Conditions of mediation

ConTeXtures are based on mediated contextures. The mediation is realized by the proemial relation. The question arises which topics can be combined and mediated. Are there limits in the combination of different topics? Can we freely combine and mediate, say numeric, symbolic, Boolean and other topics together? It has to be studied which poly-topic combinations are realizable in ConTeXtures and how they are implemented.

In this question of possible mediations of different topics and programming styles here are two situations to be considered. One is given in architectonics with commutative complexity, say situations with more than 3 contextures. In this situation there are no limits of combinations for some contextures because they are separated and thus their objects are disjunctively separated, too. Say, the topics of system1 and the topics of system4 are independent of each other because their is a "mediation gap" between them. The other situation is given with closely mediated system, there conditions of mediation are restricting the combinatorics of mediation of different topics. Say, system2 and system4 or system1 and system2 are closely connected with each other and therefore asking for strict mediation.

In ARS, and similar approaches, topics are based on general objects, called by Curry *obs*. Obs are primary to the constructed topics like Booleans and Church Numerals. Topics are not pre-given but constructed, more precisely, re-constructed by the lambda and combinatory logic operations.

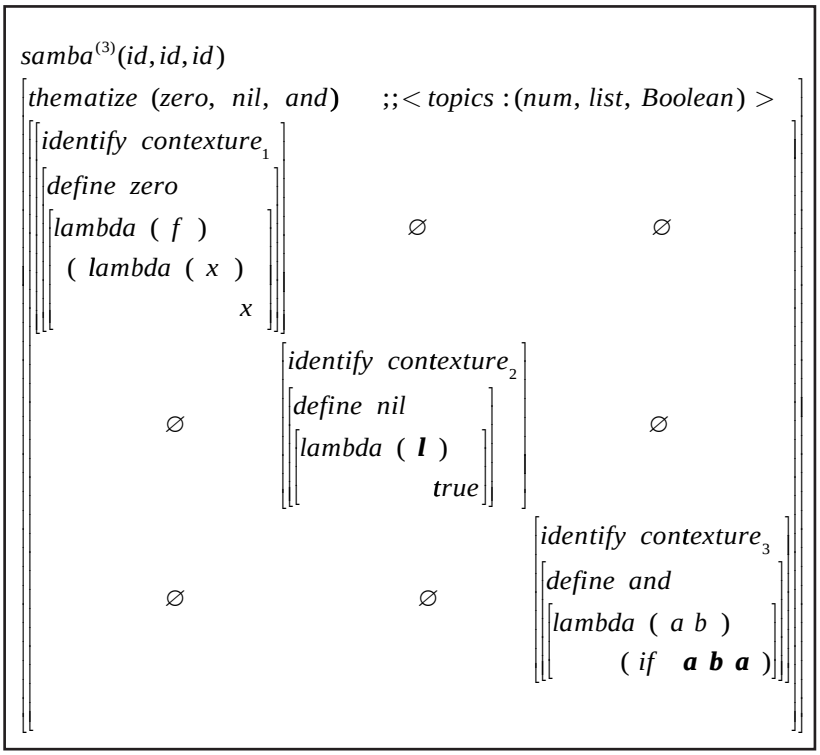
Proemiality between topics is defined by the relations of order, exchange and coincidence. To mediate singular topics over contextures the relation of exchange is of crucial relevance. It has to be guaranteed and constructed. The order relation is given intra-contexturally by the main relations and elements of the topics. The coincidence relation which is controlling the categorial adequacy of the mediation is automatically realized because of the sameness of the distributed topics. That is, Booleans distributed over different places remain Booleans and are therefore fulfilling the criteria for categorial adequacy.

For poly-topical mediations, the situation has radically changed. The topics are categorial different. And the question, how are the mediated has to be answered. The answer seems to be given by the fact that also for ConTeXtures the primary objects are not the topics but the *c-obs*, that is, the complex formal obs. Thus, the mediation of different topics is based on the mediation of the obs to c-obs.

25.1 Interactional poly-topics

Interactional poly-topics can appear inside of a programming paradigm or can be the at the base of different paradigms. Poly-topics as such are not defining a programming style but can support their definition.

$$
\begin{array}{l}
samba^{(3)}(id,id,id) \\
\left[thematize\ (poly-topics) \right. \\
\left[\begin{array}{ccc}
\left[\begin{array}{l} identify\ contexture \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array} \right) \end{array} \right) \end{array} \right] & \varnothing & \varnothing \\
\varnothing & \left[\begin{array}{l} identify\ contexture \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array} \right) \end{array} \right) \end{array} \right] & \varnothing \\
\varnothing & \varnothing & \left[\begin{array}{l} identify\ contexture \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array} \right) \end{array} \right) \end{array} \right]
\end{array} \right] \left. \right]
\end{array}
$$



This poly-topics example shows a distribution of numeric, list and Boolean topics. Topics are understood as interpretations or constructions based on the complex premial objects, c-obs. It has to be shown that this constellation of topics is realizable on the base of c-obs. Additionally, the topics num and list, with zero and nil, are unary, while the Boolean and is binary.

25.2 Reflectional poly-topics

Metamorphic reflections

Reflection can change the meaning of an object. Objects are not defined as identical obs in the sense of an ob is an ob, but by the as-category: c-ob1 as c-ob2 is c-ob3.

Reflection is using the statement defining the object and the use is defining the meaning of the object. Reflection and contemplation or introspection of an object can produce the insight that the meaning of the object under consideration is changing.

The examples shows that the reflection interpreted an object as the number zero belonging to the topic numerals. A second reflection considers the same object not as a numeral but as nil belonging to the topic of lists. Reflection has not to come to a still stand and can go further and with the interpretation and realizing that the object can be understood as belonging to the topic Booleans and appearing as the truth-value true. Therefore the object is conceived as having a numerical, a symbolic and a Boolean meaning.

Conditions of metamorphic reflections

Non-metamorphic reflection repeats the full structure of the formula over all ranks of reflections. If a n-ary Boolean function is reflected at a starting level it remains a n-ary Boolean function at all ranks of reflection (introspection, contemplation).

This could be a condition for reflectional activities.

But the idea of metamorphic reflection is offering not only to change the topics of reflection but also the structure of the reflected formula. That is, a ternary function or relation can change in the process of reflection to a binary function, e.g., if reflection is producing reduction.

At the end of this contemplation it is not impossible that reflection as a form of abstraction and thematization can jump out of the very game of the ConTeXtural framework and continues in another scenario, say morphogrammatics, reflecting the morphic aspect of the reflected situation, "morphizing" the thematizing patterns of ConTeXtures.

$$samba^{(3)}(repl, \varnothing, \varnothing) - horizon$$

$$\left[thematize\ (poly-topics^{(3)}) \right.$$

$$\left[\left[\begin{array}{l} identify\ contexture_{1.1} \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right.$$

$$\left[\left[\begin{array}{l} identify\ contexture_{1.2} \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right.$$

$$\left. \left. \left[\begin{array}{l} identify\ contexture_{1.3} \\ \left(\begin{array}{l} define\ name \\ \left(\begin{array}{l} lambda \\ \{statements\} \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right] \right]$$

Reflectional poly-topics scheme over the 3 internal contextures.

Because of its poly-topics the full head / body structure has to be distributed / replicated over the internal reflectional loci.

This guarantees the possibility of strict poly-topics in the switch from one reflectional level to the next without excluding the case of repetition of the same topic over the different places.

As an example, the internal distribution of the topics Number, List, Boolean is shown as (zero, nil, false) at the locus O1.

Thematize (zero, nil, false) is distributed internally over 3 places by the superoperator replication (repl).

There is a slight ambiguity in the notation produced by the abbreviated form, but ruled properly by replication. Otherwise we would have to write the full scheme for *thematize*:

[(zero, nil, false) (., . , .) (., ., .)]

Nevertheless it has to be proven that this constellation is not violating the structural conditions of mediation.

$$samba^{(3)}(repl, \varnothing, \varnothing) - horizon$$

$$\left[thematize\ (zero, nil, false) \right.$$

$$\left[\left[\begin{array}{l} identify\ contexture_{1.1} \\ \left(\begin{array}{l} define\ zero \\ \left(\begin{array}{l} lambda\ (\ f\ ) \\ (\ lambda\ (\ x\ ) \\ \qquad\qquad x\ ) \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right.$$

$$\left[\left[\begin{array}{l} identify\ contexture_{1.2} \\ \left(\begin{array}{l} define\ nil \\ \left(\begin{array}{l} lambda\ (\ l\ ) \\ \qquad\qquad true\ ) \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right.$$

$$\left. \left. \left[\begin{array}{l} identify\ contexture_{1.3} \\ \left(\begin{array}{l} define\ false \\ \left(\begin{array}{l} lambda\ (a\ b\ ) \\ \qquad\qquad b\ ) \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right] \right]$$

The significance of this reflectional poly-topic constellation is that what started the reflection as the number zero turns out to be on a further level of reflection, but well connected to the start, as a list topic, nil. The comparison of the reflexion of both models, the number and the list object, appears as a Boolean object, false, accepting the relevancy of all previous meanings of the objects as different topics at different places. The example is not considering neighbor systems, thus, they are empty.

25.3 Contextualizing the define operator by the AS-abstraction

Again, the as-abstraction can be defined intra-contextural, contectualizing names from context to context inside a contexture and trans-contextural, contexturalizing names between different contextures. The step forward in formalizing polycontextural programming is: From "define name" to "define a name as a name".

The steps of interpretations of names, $name_i$ as $name_j$, can be made explicit by introducing the as-abstraction in the "name giving" procedure. Thus, the innocent identifier "define name" has to be enlarged to "define $name_i$ as $name_j$"

This AS-function I have introduced as an extension of the OOP categories: IS and HAS. But it is introduced at the very beginning of the lambda construction "define name" as it appears on the base of the proemial relationship. The as-function is also crucial for the definition of metamorphic reflections.

Not only contextures can be named and names be contexturalized but also names can be named as other names between and inside of contextures in the sense that $name_j$ refers to another context or contexture than $name_i$, both belonging to the complexion involved in the play. Therefore, the referentiality and transparency of ConTeXtures is not restricted to any hierarchy of tectonics.

Levels and meta-levels of reflection are connected by means of proemiality realizing its structural rules of exchange, order and categorial correctness (coincidence) avoiding wild jumps, structurally possible, but not analyzed in this context.

$$samba^{(3)}(repl, \varnothing, \varnothing) - horizon$$

$$\left[thematize\ (zero,\ nil,\ false) \right.$$

$$\left[\left[\begin{array}{l} identify\ contexture_{1.1} \\ \left(\begin{array}{l} define\ zero \\ \left(\begin{array}{l} lambda\ (\ f\ ) \\ \left(\begin{array}{r} (\ lambda\ (\ x\ ) \\ x \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right.$$

$$\left. \left[\begin{array}{l} identify\ contexture_{1.2} \\ \left(\begin{array}{l} define\ zero_{1.1}\ as\ nil \\ \left(\begin{array}{r} lambda\ (\ l\ ) \\ true \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right.$$

$$\left. \left. \left[\begin{array}{l} identify\ contexture_{1.3} \\ \left(\begin{array}{l} define\ nil_{1.2}\ as\ false \\ \left(\begin{array}{r} lambda\ (a\ b\ ) \\ b \end{array} \right) \end{array} \right) \end{array} \right] \quad \varnothing \quad \varnothing \right] \right]$$

Thus, "define *name*" is an abbreviation of "define $name_i$ as $name_j$" with i=j.

– replication *repl,* in this example, is a metamorphic replication and not replicating isolated configurations.

Exchange relations:

– "define *zero*" is "define *zero* as *zero*", as the start of the levels. It could itself be produced by a predecessor level.

as: define *zero* in $contexture_{1.1}$ as *zero* in $contexture_{1.1}$

– "define *nil*" is "define *zero* as *nil*",

as: define *zero* from $contexture_{1.1}$ as *nil* in $contexture_{1.2}$

– "define *false*" is "define *nil* as *false*".

as: define *nil* from $contexture_{1.2}$ as *false* in $contexture_{1.3}$

This change of identity of the topics from one contexture to another by reflection / replication is producing a chiastic chain guarantying the connectedness of the step-wise reflection of the whole. A neighbor system, S1 or S2, could reflect in its contexture this fact of chiastic connectedness.

G. Dissemination of Programming Styles

The minimal condition for the dissemination of different programming paradigms is given by the poly-contextural matrix. Different paradigms have to be placed, they have to take place and to realize themselves at an epistemological place.

26 How are programming paradigms defined in ARS?

26.1 Object-oriented programming style in ARS

It is beyond the scope of this study to go into the details of the definition of the OOP terminology on the base of ARS. This can be found online at the A++ web site.

The emphasis is on constructors:

> *Constructors*
> In most object oriented programming languages, classes are defined by special constructs and the constructor generates an object according to the class definition. These languages can also be labelled `class oriented'.
>
> In A++ however, classes are defined entirely by constructors, which like in other languages creates objects as instances of the specific class.
>
> *Components of a constructor*
> • input arguments
> The input arguments passed to a constructor are used to initialize the attributes.
> • definition of the attributes
> Normally all attributes are defined in A++ by performing an abstraction in the first of the two possible forms, i.e. using the key word `define'.
> • definition of the methods
> The methods are of course lambda-abstractions and must be given a name using `define'.
> • definition of the special method `self'
> This method is a very special lambda-abstraction, because it is returned by the constructor as the closure representing the entire object. As closure being defined and contained in the constructor it has access to everything in the constructor, i.e. all attributes and methods. This is why `self' can serve as a dispatcher, receiving the messages sent to the object, interpreting them and mapping them to calls to the corresponding methods.
> • return value
> The return value of the constructor is the lambda-abstraction `self' as described above.

http://www.aplusplus.net/bookonl/node57.html

26.1.1 SELF at the edge of a self-referential calculus

The method 'self' allows to introduce the distinction between intra-contextural and trans-contextural self-referentiality. ARS is thanks to the method 'self' involved in self-referentiality which seems not to be usual for programming concepts. This ARS is in its tectonics not strictly hierarchic but has important features of circularity in the sense of reflection-into-itself without producing contradictional conflicts.

It is only one more step in thematizing the power of *self* if it is allowed to self-thematize its own frame, which is ARS as a whole. ARS as itself an object. Instead of putting the reflected ARS into itself it can be localized, positioned on a own locus, different to the reflected system ARS opening up the possibility of mediation of the distributed systems. The method 'self' is therefore not restricted to a dispatcher but open to distribution.

26.2 Imperative programming style in A++

The abstraction `while'

It is remarkable, that such a 'while'-construct can be defined in A++ as a lambda-abstraction. In most of the other programming languages such an attempt would be in vain.

It should be noted however that all the statements to be repeated are bound together by one `lambda'. This is necessary because `while' is defined as a lambda-abstraction expecting two arguments. One argument stands for the condition that controls whether another iteration should occur and the second variable stands for the block to be repeated. The possibility to define lambda-abstractions with a variable number of arguments is not implemented in A++ right now. http://www.aplusplus.net/bookonl/node172.html

```
(define while
  (lambda(c body)
    (define loop
      (lambda()
        (if c
            ((lambda()
               (body)
               (loop)))
            false)))
    (loop)))
;
(define while-test
  (lambda(n)
    (while (gtp n zero)
      (lambda()
        (define n (pred n))
        (ndisp! n)))))
```

```
[define while
([lambda ( c body )
([define loop
 ([lambda ( )
 ([if c
  {[lambda ( )]
   [( body )]
   [( loop )]}
  false])])
( loop )])])]
```

26.3 Games of inscribing programming styles

Diagramm 14 **Hierarchical foundation of programming paradigms**

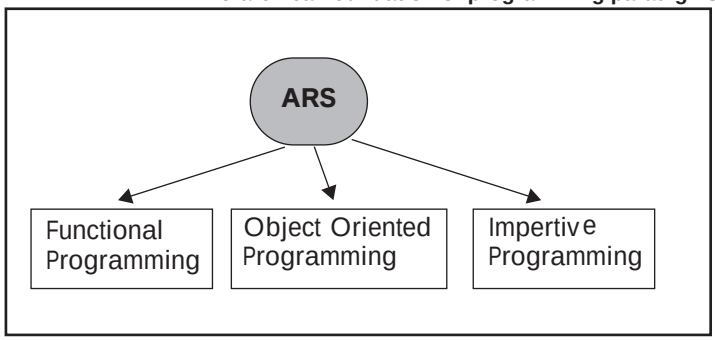
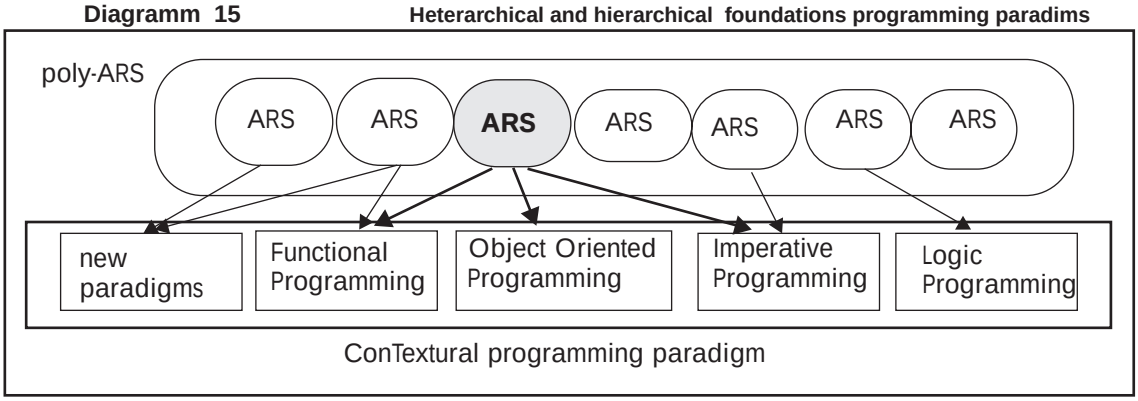
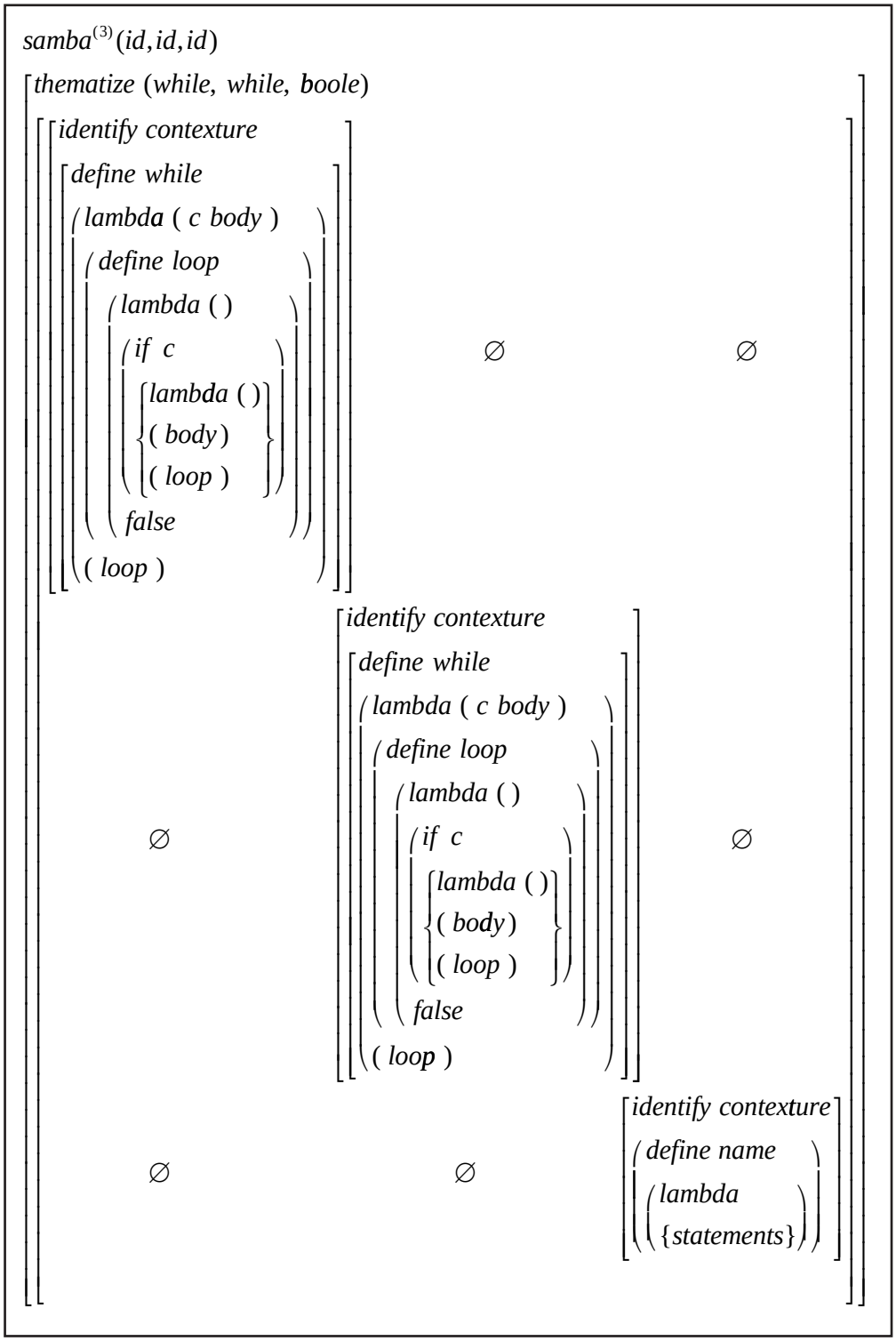
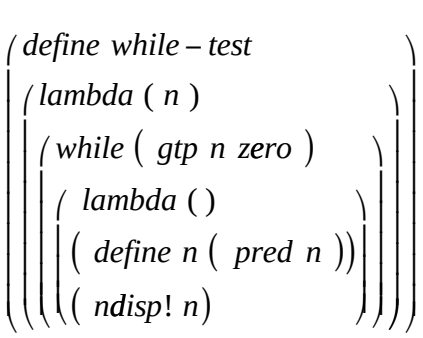


Diagramm 15 **Heterarchical and hierarchical foundations programming paradims**





The example shows the distribution of the while-construct over two contextures in the mode of computation at the places of the contextures. That is, without interaction and reflection between contextures.

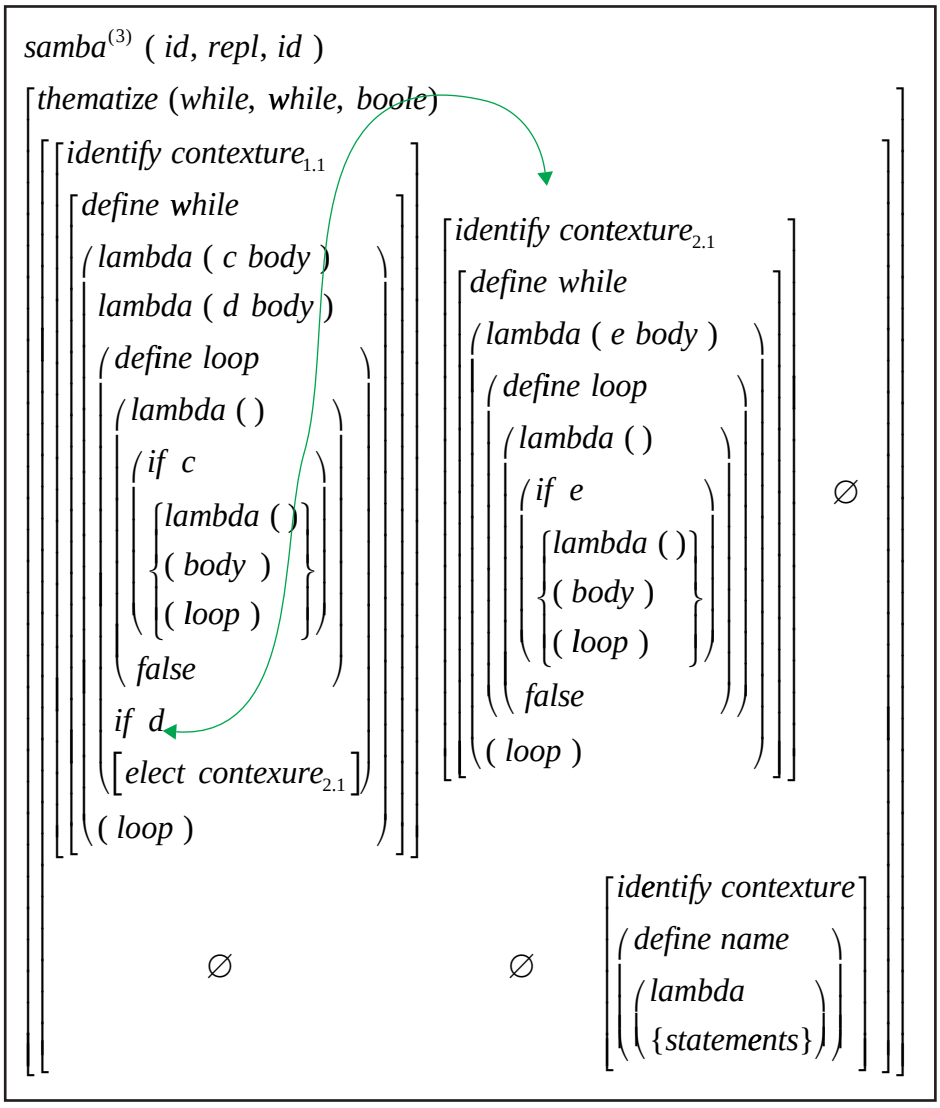


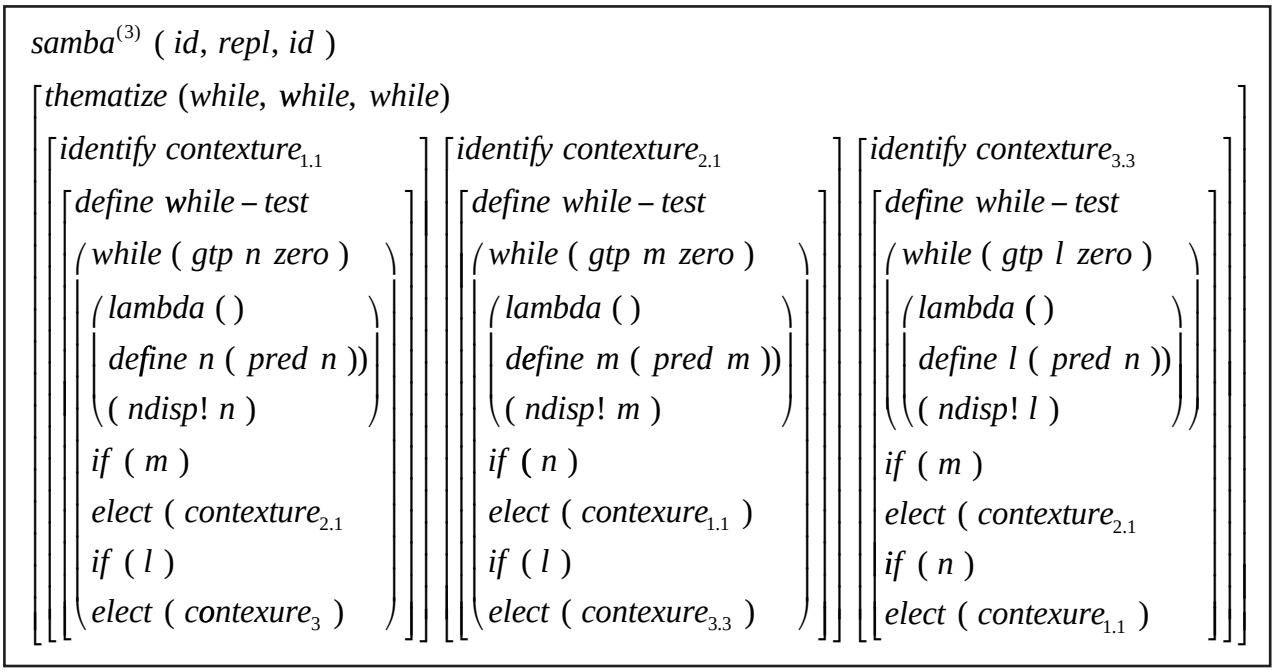
This is the while-test for a single loop involved at a place of a contexture. .

What could it mean to distribute the while-construct in an interactive and a reflectional manner?

Each loop starts with the decision made by the elector "*elect*" to stay in the contexture of the construct or to leave the contexture of the construct and to switch to another contexture which has realized the same while-construct or another while-construct or even something different. This is realized by the double function of the operator "*if*". The conditional *if* includes the intra-contextural selector "*sel*", but in polycontextural environments the function of selection can change to the function of election (of a new contexture). This jump is not yet implemented in our first example of the distribution of the while-construct over two contextures, but the conditions for the jump are given.

What we need now is a new condition *(if d [elect contexture2.1])*, which forces the selector to decide the procedure to leave the loop, to let it jump and continue in another contexture. In the example, the jump leads from contexture1.2 to contexture2.1. The third contexture3.3 may reflect this mechanism between both contextures. It could reflect if this jump happens or not, thus it could be realized by Boolean functions.


samba(3) (id, repl, id)
thematize (while, while, boole)
identify contexture1.1
define while
lambda (c body)
lambda (d body)
define loop
lambda ( )
if c
lambda ( )
(body)
(loop)
false
if d
elect contexture2.1
(loop)
identify contexture2.1
define while
lambda (e body)
define loop
lambda ( )
if e
lambda ( )
(body)
(loop)
false
(loop)
∅
∅
∅
identify contexture
define name
lambda
{statements}


samba(3) (id, repl, id)
thematize (while, while, while)
identify contexture1.1
define while – test
(while (gtp n zero)
(lambda ( )
define n (pred n))
(ndisp! n)
if (m)
elect (contexture2.1
if (l)
elect (contexure3)
identify contexture2.1
define while – test
(while (gtp m zero)
(lambda ( )
define m (pred m))
(ndisp! m)
if (n)
elect (contexure1.1)
if (l)
elect (contexure3.3)
identify contexture3.3
define while – test
(while (gtp l zero)
(lambda ( )
define l (pred n))
(ndisp! l)
if (m)
elect (contexture2.1
if (n)
elect (contexure1.1)

27 OOP style programming in ConTeXtures

27.1 Root and self

- definition of the special method `self'

  This method is a very special lambda-abstraction, because it is returned by the constructor as the closure representing the entire object. As closure being defined and contained in the constructor it has access to everything in the constructor, i.e. all attributes and methods. This is why `self' can serve as a dispatcher, receiving the messages sent to the object, interpreting them and mapping them to calls to the corresponding methods.

http://www.aplusplus.net/bookonl/node61.html

As it is well known, OOP is a highly hierarchical enterprise.

To give introduce first steps to a heterarchic structuring of OOP we can reduce our efforts to a deconstruction of the operator "self" and its connection to the "base-object" which is the root-object of the hierarchy.

As introduced in DERRIDA'S MACHINES there are good reasons to heterarchize hierarchically organized systems.

27.2 Relation between classes

The duality of relations below corresponds to the duality `inheritance' and `delegation' we have just discussed.

- **IS-Relation**

  This relation has to do with inheritance: A dog is an animal. A car is a vehicle. Because a dog is animal it has all the features of animal and of course on top of that its own.
- **HAS-Relation**

  In the system using delegation described above we have seen that an object must have an instance of its super-class in its attributes in order to be able to delegate an unknown message to a higher level.

  Of course an object may have instances of several super-classes making it easy to implement multiple inheritance.

http://www.aplusplus.net/bookonl/node69.html

The IS- and HAS-relations are surely intra-contextural relations and all their consequences for the programming system, too.

Problems with IS- and HAS-relations

All that sounds well established and not entangled with any problems which could lead to a questioning of the concepts involved. But as usual things are not so simple at a second glance. Two largely unsolved main problems are well known: *polysemy* and *multiple inheritance*.

The known strategies to deal with polysemy are all leading to an infinite regress.

The proposed solutions to multiple inheritance are leading, taken seriously, to paradoxes and contradictions.

As a first step to enhance the classical concept of ontology which is behind the OOP approach we should introduce a new kind of relations: *AS-relation*.

- **AS-Relation**

An object X, thematized as an object Y, is an object Z.

The statement "A dog thematized as a dog is a dog" is a realization of the form "X as X is X".

It is easy to understand, that the classic ontological identity formula "X is X", which is used as the base of "everything", is an abbreviation of the reflectional form "X as X is X".

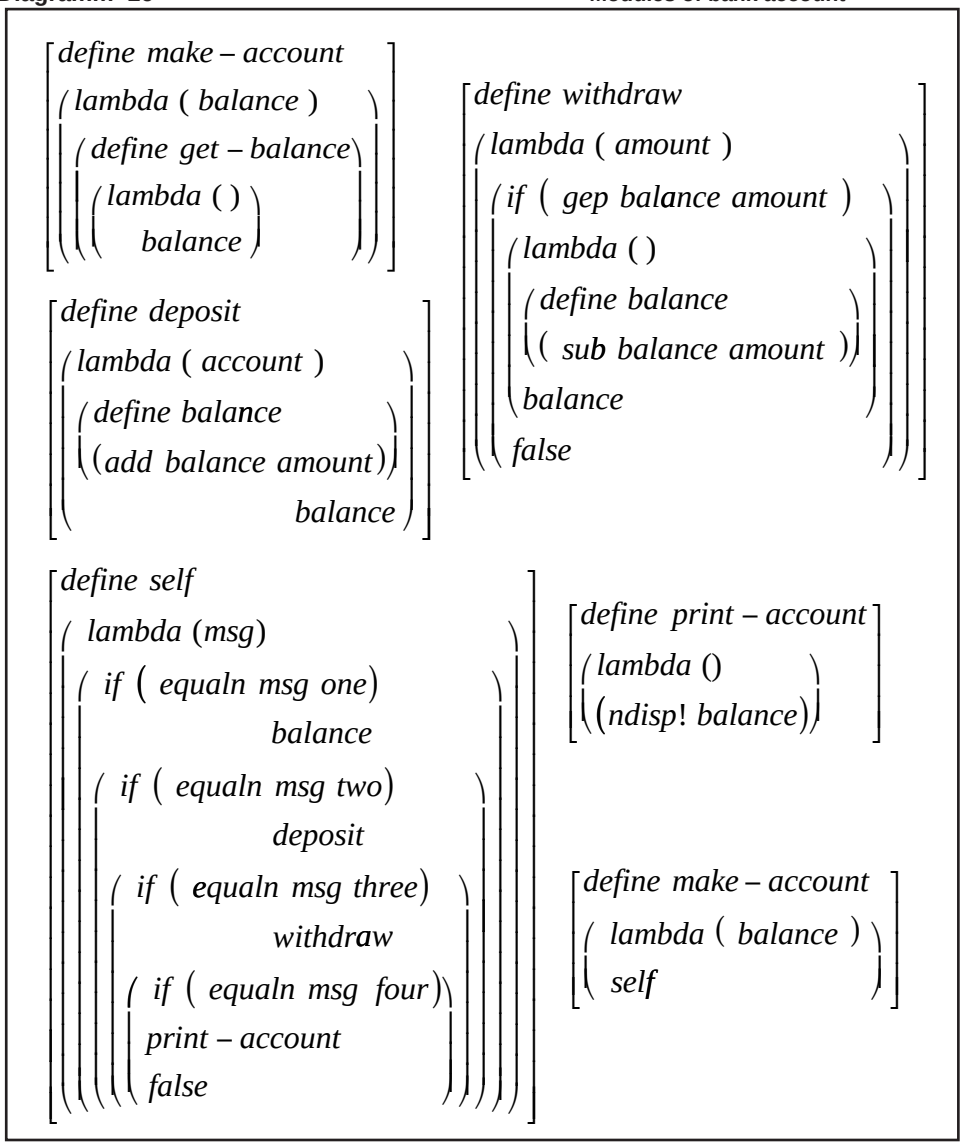
This classic-ontological way of thinking is guiding everything in computer science and programming and is not restricted to the case of OOP. But in practice, the real life of programming looks quite different. It starts even at the very beginning of computing with the violation of its ontological principle. Take the use of programs AS data and the use of data AS programs. Or the interpreter as a program and the program as an interpreter. No problem! Yes, but also no theory of this practice.

An animal shelter is an animal shelter. And we got the program to define the behavior of it. But an animal shelter as a (temporary) shelter for fighting soldiers is surely not an animal shelter anymore. Not only by the substitution of the reference but also by its functionality. So we need a new program "soldier shelter" and then, because this was only of temporary use, a new program for the transitions between both functionalities, the animal and the soldier shelter. But we can also have the situation of a simultaneity of both, at once animal and soldier shelter. And so on.

http://www.aplusplus.net/bookonl/node142.html

27.2.1 Bank account as a simple example

Diagramm 16 **Modules of bank account**



27.2.2 Analysis and Comments

Why should we heterarchize this bank account model?

As we can see, all the modules are quite autonomous defined and embarrassed only by the clam of self which is organizing the modules / objects into a hierarchic order to function as a OO program.

One reason could be the constellation of many different accounts belonging, say to the same user. For each account the particular objects with their specific attributes has to be defined and put together in some kind.

The accounts can differ and change over time on all parametric levels: currency, type of printing, etc. In a extreme situation they can even differ in their arithmetics and logic of the accounts, and nevertheless they have to cooperate and work as a complex account.

In a heterachic setting, new features and new modules can easily added and old modules can put to sleep without costing the system anything.

This can be done more efficiently than in re-programming the users accounts into one big new OO constellation. Anytime the user is changing the attributes or the number of accounts this procedure has to be repeated.

This kind of flexibility of modularization surely was a main attempt of OOP.

ConTeXTures differs from OOP in choosing additionally to the hierachic organization of the objects the new organizational dimension of *heterarchy* which deliberates the modules from their hierarchic encloser and offers space for more autonomy and interactivity with the involvement of contextures.

In other words: the root or self is simply an object next to other objects.

Ontologically speaking this says that the whole is taken as a part in another contexture.

The connection between different objects in different contextures is not a link but an interaction between contextures.

Even a single class, here account, can be put into a heterarchic setting reflecting its intrinsic structure. This may lead to data based parallelism.

Diagramm 17 **Metapattern of bank account**